871780 — MonB5G — ICT-20-2019-2020

Deliverable D3.2
Final Report on AI-driven Techniques for the MonB5G AE/MS

Document Summary Information

Grant Agreement No	871780	**Acronym**	MonB5G
Full Title	Distributed Management of Network Slices in beyond 5G		
Start Date	01/11/2019	**Duration**	42 months
Project URL	https://www.monb5g.eu/		
Deliverable	D3.2 – Final report on AI-driven techniques for the MonB5G AE/MS		
Work Package	WP3		
Contractual due date	M30	**Actual submission date**	30.04.2022
Nature	Report	**Dissemination Level**	Public
Lead Beneficiary	NEC		
Responsible Author	Zhao Xu (NEC)		
Contributions from	Bahador Bakhshi (CTTC), Luis Blanco (CTTC), Engin Zeydan (CTTC), Hatim Chergui (CTTC), Josep Mangues (CTTC), Jordi Serra (CTTC) George Tsolis (CTXS), Adlen Ksentini (EUR), Marina Costanti (EUR), Sabra Ben Saad (EUR), Thrasyvoulos Spyropoulos (EUR), Anestis Dalgkitsis (IQU), Luis A. Garrido Platero (IQU), Anne-		

	Marie Bosneag (LMI), Ashima Chawla (LMI), Zhao Xu (NEC), Sławomir Kukliński (ORA-PL)

Revision history

Version	Issue Date	Complete(%)	Changes	Contributor(s)

Disclaimer

Copyright message

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	*Description*
3GPP	*Third Generation Partnership Project*
AE	Analytic Engine
AE-F	Analytic Engine Function
AE-S	Analytic Engine Sublayer
AI	Artificial Intelligence
APM	Application Performance Monitoring
CLA	Closed-loop Automation
CNCF	Cloud Native Computing Foundation
CNF	Cloud Native function
DE	Decision Engine
DE-F	Decision Engine Function
DE-S	Decision Engine Sublayer
EEM	Embedded Element Manager
eMBB	Enhanced Mobile Broadband
ETSI	European Telecommunications Standards Institute
ECA	Event Condition Action
ENI	Experiential Networked Intelligence
FCAPS	Fault, Configuration, Accounting, Performance, Security
InP	Infrastructure Provider
ISM	In-Slice Management
ITU	International Telecommunication Union
KPI	Key Performance Indicator
LCM	Lifecycle Management
ML	Machine Learning
MANO	Management and Orchestration
MaaS	Management as a Service
MAN-F	Management Function

mMTC	Massive Machine Type Communications
MEO	MEC Orchestrator
MNO	Mobile Network Operator
MLaaS	MonB5G Layer as a Service
MS	Monitoring System
MS-F	Monitoring System Function
MS-S	Monitoring System Sublayer
MEC	Multi-access Edge Computing
NFVO	Network Function Virtualization Orchestrator
NSD	Network Service Descriptor
NSO	Network Service Orchestrator
NSP	Network Service Provider
NSI	Network Slice Instance
NSMF	Network Slice Management Function
NSSMF	Network Slice Subnetwork Management Function
NST	Network Slice Template
NSSI	Network sub-Slice Instance
NGMN	Next Generation Mobile Networks
NFVI	NFV Infrastructure
OAI	Open Air Interface
ONAP	Open Network Automation Platform
OSM	Open-Source MANO
OSS	Operation System Support
PaaS	Platform as a Service
PoC	Proof of Concept
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
SON	Self-Organizing Network
SLA	Service Level Agreement

SFL	Slice Functional Layer
SML	Slice Management Layer
SM	Slice Manager
uRLLC	Ultra-Reliable Low-Latency Communication
VIM	Virtual Infrastructure Manager
VNF	Virtual network Function
VNFM	Virtual network Function Manager
ZSM	Zero-touch network and Service Management

1 Executive summary

Due to the huge business potential brought by the novel network slicing technology, a large number of slices will co-exist in the 5G/B5G infrastructure. The traditional centralized management mechanism will suffer greatly from the intensive communication between the central administration and the functional entities, especially when the number of slices increases and they are deployed in multiple technical domains. The delays caused by inefficient management will significantly degrade the Quality of Service (QoS) and even violate Service Level Agreements (SLAs) with customers.

MonB5G intends to deploy a novel autonomous management and orchestration framework to address a critical challenge in 5G and beyond, namely managing a massive number of network slices with different requirements and functions. This deliverable reports the developed Monitoring System (MS) and Analytics Engine (AE), two essential management entities of the MonB5G platform, towards Zero-Touch Scalable Slice Management and Orchestration. They leverage distribution of management operations along with state-of-the-art AI-based mechanisms for scalability, efficiency and automation. The developed system is based on a hierarchical approach that enables flexible and efficient management of network tasks while introducing a diverse set of decentralized levels through optimal adaptive assignment of monitoring and analysis functions for the current and future status of slices.

The major technical contributions and innovations of MS/AE focus on *Scalability* and *Automation*. We embed MS/AE into slices as management entities that provide slice-specific monitoring and analysis services. Within each slice, the hierarchical management strategy is extended and applied. Monitoring and analysis operations are distributed to the local management entities associated with the corresponding functional units such as VNFs, domains and slices near which the management data and tasks are generated. The distribution of operations between adjacent layers and between entities on the same layers is optimized using AI based techniques such as federated learning. The state-of-the-art machine learning and representation learning techniques are tailored and developed for the decentralized management tasks. The resulting MS/AE enables online monitoring and analysis of each layer of functional entities for a massive number of slices with a significant reduction in communication overhead and delay.

In particular, the MS is based on the specifications for autonomic network management and cloud-native design. The MS can be conceived as a cross-domain virtual layer hosted by a NFV IFA 029 compliant PaaS (i.e., Container Infrastructure System (CIS)). The decentralized MS distributes monitoring tasks across multiple levels of the management hierarchy (node, slice, domain, and inter-domain) in a programmable manner. After being triggered and configured by AE, the programmable MS entities connect the corresponding infrastructure and functional entities, e.g., VNFs, to gather the requested telemetry data with the granularity defined by AE. We introduce three types of APIs, including Control API, Data Collection API, and Data Processing API, to connect the MS entities to a MS bus to process real-time data feeds for unified, high-throughput and low-latency communication. As shown in Figure 1, MS provides the following key benefits:

- The distributed MS entities, as part of the Slice Management Layer (SML), fulfil tight metrics sampling loops (see Sampling Function, abbreviated as SF, in the figure) for the entities in the Slice Function Layer (SFL) where the management data/tasks are generated, so that the communication overhead introduced by MS itself is minimized.

- Additional MAPE-based embedded element managers (EEMs) are deployed with the functional entities to support fine-grained local telemetry collection. It also provides additional flexibility of specific aggregators for various AE and DE functions.
- The configurations of the distributed MS entities are defined and triggered by the AE/DE entities with AI-based policy-driven mechanisms that represents a decisive step towards a highly automated slice-level monitoring system.

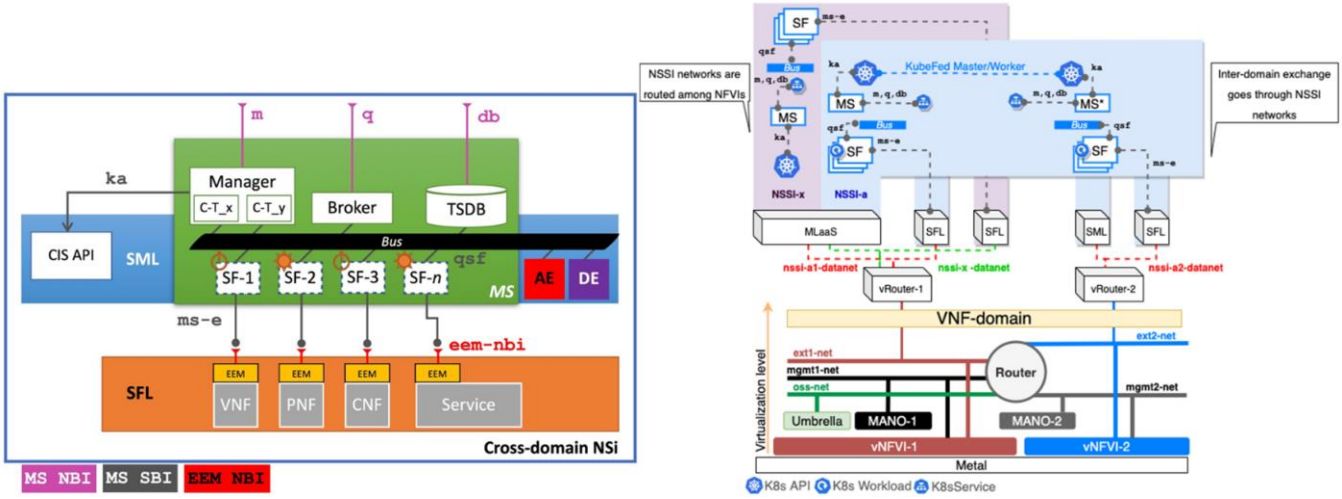


Figure 1 Illustrates the MonB5G MS: internal components and reference points (left) and implementation of MS for two domains (right)

AE provides a variety of slice-specific analytics for inter-domain, cross-domain and network-aware KPI inspections with a focus on AI-based scalability and automation. The novelty of AE focuses on the challenges posed by decentralized management. On the one hand, AE distributes analytics tasks locally to reduce communication overhead and unexpected delays. On the other hand, local analysis lacks a global view of slice/network status, so predictions can be less accurate and even deviate from the actual situation. Moreover, an AI driven strategy should be used to learn how to optimally distribute analysis operations among management entities. To this end, the techniques of distributed ML and federated learning are being explored. The main innovations are shown in Figure 2. In this process, abstract information (patterns and data representations) is extracted and exchanged between lower and upper layers of the management hierarchy to balance local predictive performance with time/resource costs. In addition, other novel ML techniques, such as representation learning and context-aware analytics, are being explored for AE. The main AE features include, for example:

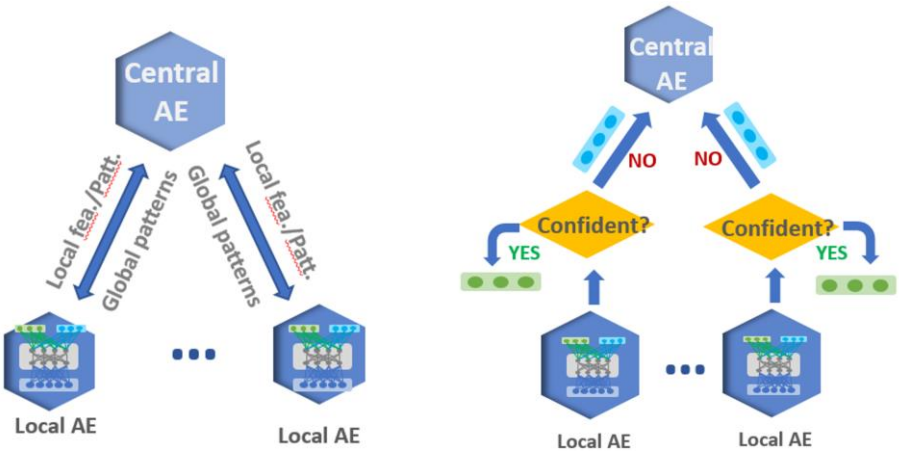


Figure 2 Decentralized AE operations via federated learning (left) and distributed neural networks (right)

- Federated Resource estimation with low SLA violation: This AE feature introduces a set of well-designed statistical constraints for distributed network management with enhanced federated learning. The novel feature facilitates decentralized resource allocation in network slices while guaranteeing very low violation of SLA.
- Moving analysis locally with distributed neural networks: The distributed AE entity can support slices that are deployed in different domains. It facilitates prediction of sophisticated KPIs that depend on the performance of all components (VNFs, links, across domains) of a slice. The prediction offloading mechanism is data-drive, learn to optimize itself based on actual situations.
- Enhanced traffic load prediction: Traffic load forecasting is essential for many downstream tasks such as resource allocation and admission control. The innovative AE feature can predict traffic load for any technological domains. This feature integrates additional regularizations to model penalties for over and under allocation of resources as well as resource reallocation settings. By ensuring that the right number of resources are provisioned to a network slice when needed, it significantly reduces the likelihood of SLA violations, and guarantees user perceived QoS.
- Local fault detection enhanced with neighbourhood information: This AE feature identifies the local anomalies based on the learned normal behaviours of the monitored entities. By employing a graph-based neural network, we integrate the information of neighbourhood entities. The anomalies can thus be better detected, as this feature considers not only the status of the monitored entity itself, but also a globe view of the related entities.
- Cross-domain anomaly detection with distributed optimization: All nodes in the network can obtain the global minimum value by communicating only with their neighbours, without the need of a central coordinator. The AE feature is asynchronous, namely, nodes can activate at any time without having to wait for any other specific event in the network. Moreover, this feature does not require data exchange among nodes, which largely reduces communication overhead.

In summary, MonB5G MS and AE enable distributed AI-driven monitoring and analytics for managing a large number of slices. The decentralized monitoring and analytics operations are distributed among the hierarchically structured management entities with a focus on reducing communication overhead and delay. The distributed ML and FL techniques are extended to fit the decentralized management architecture, i.e.,

they automatically learn the distribution policies to balance predictive performance and overhead/delay. This work is an important step towards scalable intelligent monitoring and analytics capabilities for zero-touch slice management.

2 Introduction

2.1 Scope

This deliverable reports on the accomplishments of Work Package 3 (WP3) of the MonB5G project related to AI-driven techniques for the MS and AE for Zero-Touch Distributed Slice Management and Orchestration. Beyond the initial development reported in the deliverable D3.1, this document covers the improved design and implementation of MS and AE and the promising validation results. In particular, we explore the scalability of the components, with distributed AI techniques tailored to the decentralized and programmable management architecture leading to significant reductions in communication overhead and time delays that can result from monitoring and analysis. We also report on two datasets generated with 5G simulators that were used to test the proposed analysis functions, and will be published as an outcome of the MonB5G project for the benefit of the 5G/B5G research community.

To the best of our knowledge, the proposed MS and AE are the first attempt in the literature that can fully embed themselves in a decentralized management platform, significantly increasing the efficiency and scalability by distributing these management operations to adjacent layers and multiple entities in the same layer. The AI-driven techniques are explored to learn the optimal distribution policies considering the highly dynamic network status. Although the operations are performed locally, near the nodes where monitoring telemetries are generated and orchestration is performed, the use of the distributed AI techniques still enables a comprehensive and accurate analysis that integrates a global view of the network entities involved. Overall, the resulting MS and AE provide promising enablers for monitoring and analyzing a massive number of coexisting slices using distributed, AI-driven techniques with low-overhead and delay.

2.2 Motivation

Given the huge benefits of network slicing for Mobile Network Operators (MNOs) and their customers, 5G networks are expected to support a massive number of coexisting network slices with different performance requirements, functionality and time spans [1] [2] for a variety of vertical industries. The commonly used centralized network management systems face significant challenges to handle the highly complex networks. An intelligent and scalable slice management platform is expected to orchestrate network infrastructure and resources to dynamically and proactively enable the requested services and preferences. To this end, the MonB5G platform has been developed with promising achievements.

As important management entities of the MonB5G platform, MS and AE will track and analyse the current and future status of slices and participating functional entities deployed in various technical domains. The telemetry data collected by MS and the analysis results from AE are important indicators that will be provided to Decision Engine (DE) to optimize admission and orchestration. Since a large number of slices coexist in the network infrastructure and each slice is often deployed in different technical domains, some new challenges arise. Technically, the MS must meet the following requirements:

- The centralized cloud management system will need to evolve to a distributed, network state-aware system to handle large number and high dynamicity of slices envisioned in 5G scenarios and beyond. This will improve both the scalability and reaction time of network slice self-management and self-

configuration to achieve true zero-touch network management. Deploying such a platform will require effective, detailed and sophisticated monitoring of KPIs and subsystem behaviour metrics, the analysis of which will reveal potential or novel issues in the functionality of the framework.

- A distributed management plane must go hand in hand with the use of data-driven mechanisms based on Artificial Intelligence (AI) algorithms for both distributed data analytics and automated decision making and optimization. For AI-driven implementations of these components to be able to respond automatically, rapidly, and scalably to non-stationary network conditions, new traffic patterns, and evolving slice characteristics and intent policies, novel distributed Machine Learning (ML) algorithms are needed. Training these algorithms requires the collection of large datasets of system-level information. Such information can only be obtained through a cutting-edge monitoring system, the deployment of which is an overall priority for MonB5G.

As for AE, it faces a number of other technical challenges. To support automatic data analytics for large number of 5G network slices, the AE will use novel ML/AI techniques. Current research results have demonstrated the potentials of ML/AI. However, slice-based network management raises a set of new technical issues in managing heterogeneous resources (e.g., communication, computational and storage). Previous H2020 calls have already established a solid framework for developing uniform network management and orchestration systems, (e.g., 5G!Pagoda, 5G-EVE, 5GENESIS and SliceNet) but there are still open issues to enable intelligent, scalable, and proactive slice analytics solutions, such as:

- Distributed management plane to support massive deployment of network slices.
- Definition of novel end-to-end (e2e) slice Key Performance Indicators (KPIs) and development of AI-based mechanisms for their accurate prediction from multi-level metrics.
- Data-driven management system based on federated learning.

To solve all these challenges, we are developing MonB5G MS and AE with promising results. They are embedded in the slices as distributed AI-based management entities that are deployed locally in different technical domains, but collaborate to automatically monitor and analyse the corresponding slices. The distributed ML and federated learning methods are developed to accurately analyse the selected slice KPIs for resource-efficient decentralized management.

2.3 State-of-the-Art Technologies and Specifications

Data collection and analytics for 5G networks have been investigated extensively in both research literature and standardisation, as they play an important role in network (slice) management. This section provides a brief review of state-of-the-art techniques as well as the recent releases from 3GPP related to these entities. Further details can be found in [3].

2.3.1 DATA COLLECTION (MONITORING)

Different monitoring approaches are being researched and proposed depending on different layers and nodes of 5G networks. For example, VNFs, deployed as VMs, leverage the telemetry capabilities of first the VIM and later NFVO/MANO stack of choice. This has led to a number of relevant projects, for example:

- OpenStack: the set of projects under OpenStack Telemetry, with Ceilometer being the one most widely adopted[1].
- OPNFV: the Barometer project[2] and the VES project[3].
- OSM: the OSM MON module and respective Performance Management capabilities[4].
- ONAP: the Data Collection Analytics and Events (DCAE) project[5].

Recently there have been efforts to enable network monitoring in a programmable fashion[6] and ongoing standardization activities under IETF[7].

On the CNCF (Cloud Native Computing Foundation) side, there is a separate set of projects under the Observability & Analysis section of the landscape[8], with Prometheus[9], fluentd[10] and Jaeger[11] as the graduated monitoring, logging and tracing projects correspondingly, with OpenMetrics/OpenTelemetry aiming to establish open standards and protocols. The open APM (Application performance monitoring) ecosystem is even broader[12].

In addition, since 5G service implementations generally adopt cloud-native approaches, service infrastructures/frameworks are expected to offer monitoring as common baseline capability. On the other hand, specialized appliances are expected to better position themselves in a hybrid multi-cloud world with cloud-native applications and services. The enhancements towards cloud native and PaaS are discussed in ETSI IFA029, where the concept of VNF common and dedicated services has been introduced. These VNFs are instantiated within the PaaS and provide data collection capabilities that are consumed by the network services running over the PaaS [4]. Importantly, a *generic monitoring service* is now considered a specific example of a VNF Common Service. Since Kubernetes is used asservice orchestration framework, the implementation will most likely be based on the technologies/projects in the corresponding area of the CNCF landscape. For example, ONF Edge Cloud[13] platforms, i.e., Aether, CORD & XOS, have already adopted the pattern of offering logging and monitoring as platform microservices, leveraging projects from the CNCF observability and open APM ecosystems (Kafka, Prometheus/Grafana and ELK/Kibana).

---

[1] https://wiki.openstack.org/wiki/Telemetry

[2] https://wiki.opnfv.org/display/fastpath/Barometer+Home

[3] https://wiki.opnfv.org/display/ves/VES+Home

[4] https://osm.etsi.org/wikipub/index.php/OSM_Performance_Management

[5] https://wiki.onap.org/display/DW/Data+Collection+Analytics+and+Events+Project

[6] https://p4.org/p4/inband-network-telemetry/

[7] https://datatracker.ietf.org/doc/draft-ietf-opsawg-ntf/

[8] https://landscape.cncf.io/category=observability-and-analysis

[9] https://prometheus.io

[10] https://www.fluentd.org

[11] https://www.jaegertracing.io

[12] https://openapm.io

[13] https://www.opennetworking.org/onf-edge-cloud-platforms/

As cloud-native and edge-enabled service deployments become a reality, the next challenges are analysing the vast amounts of telemetry data generated by monitoring systems and the need for human-in-the-loop operations which increases toil (and costs). The evolution of monitoring and APM towards greater automation and intelligence through ML/AI techniques is commonly referred to as "AIOps". The recent project Acumos AI [https://www.acumos.org] which is an integration of ONAP DCAE with the Linux Foundation, is exactly a development in this direction.

2.3.2 *DATA ANALYTICS*

Data Analytics have become increasingly important in 5G network management. A variety of ML techniques have been successfully extended and applied to provide diverse analysis functions in different problem domains.

Supervised Learning, for example, is one type of ML methods where the problems of classification and prediction are very important. The methodology for applying this type of learning requires data that is labelled in some way and whose expected outcome is known to train a function approximator that can be used to label/classify/predict the outcome of never-before-seen data inputs. Unsupervised Learning is in similar direction, but in this type of ML the labels of the data and the expected output are not known, and the function approximator must determine these labels itself, e.g., Principal Component Analysis (PCA). Another family of ML methods widely used in network management is Reinforcement Learning (RL), in which an agent learns to perform an optimal action based on its inputs and a reward function that provides the agent with a measure of its performance. There are various promising lines of research being explored for network analytics, such as:

- Traffic prediction within the context of mobile communications. The ML techniques have been successfully used to forecast traffic load of networks, slices, and VNFs [5] [6] [7] [8]. For example, Feed-Forward Neural Networks (FFNNs) and Autoregressive Integrated Moving Average (ARIMA) methods have similar performance for time-based prediction [9] [10], while LSTM-based predictors have shown better performance than ARIMA and FFNNs [11].
- Classification and Anomaly Detection. There are many types of anomalies that can be identified in a mobile communication network. These anomalies range from the simplest faults, such as links broken at the physical layer (broken wires or antennas), to application layer malfunctions and system overloads. A typical anomaly is network congestion, which has been studied using time series forecasting [12] [13] [14] [15] [16] [17]. In addition, Xie et al. [18] used a Deep RL approach to solve the congestion problem by determining the initial congestion window (IW).
- Graph Representation Learning. Graph data is pervasive in 5G networks such as the graphs created by the VNFs and virtualized links of a network slice. To handle this kind of data, diverse graph-based ML methods have been explored, e.g., Graph Convolutional Networks (GCN) [19] and Relational Graph Convolutional Network (RGCN) [20]. The representation of graph features extracted by a graph model has been successfully used in recent works on automatic virtual network embedding, e.g., [21] [22] [23].
- Federated Learning (FL). FL is a recent ML paradigm, which allows multiple agents to train a shared model in a decentralized manner without exchanging their local data. Current research on FL addresses many challenges caused by the standard centralized learning mode, e.g., [24] [25] [26] [27].

- RL for Dynamic Resource Allocation (DRA). DRA is enhanced with RL techniques to avoid or largely alleviate the problems of under- and over allocation of resources. The traditional DRA approach often sets a small utilization threshold. Resources are scaled vertically or horizontally when the predefined threshold is reached. RL methods observe the current environment – which comprises everything outside the agent and dynamically (globally or locally) select optimal actions [28]. For example, [29] develop a centralized online DRL-based dynamic resource allocation scheme for networks slicing, while the authors in [30] design and implement an adaptive online DRL approach, NFVdeep, to automatically deploy Service Function Chains (SFCs) to respond to various QoS requests.

2.3.3 *STANDARDISATION RELATED TO MS/AE*

3GPP has recently released multiple specifications related to MS and AE, such as Network Data Analytics Function (NWDAF) and Zero-Touch Service Management (ZSM). They have also been defined for dealing with network slicing. We present here the relevant functions for 5G networks and for 5G network slicing.

Network Data Analytics Function (NWDAF)

The NWDAF function was introduced in R15 (TS 23.501 [31]) by 3GPP for the 5G Core network. The details of NWDAF are described in TS 23.288 [32] and TS 29.520 [33]. The NWDAF allows network operators to either implement their own ML based data analytics methodologies or integrate third-party solutions into their networks. The NWDAF, as defined in TS 23.503 [34], is used for centralized data collection and data analytics. Figure 3 illustrates the architecture of the NWDAF:

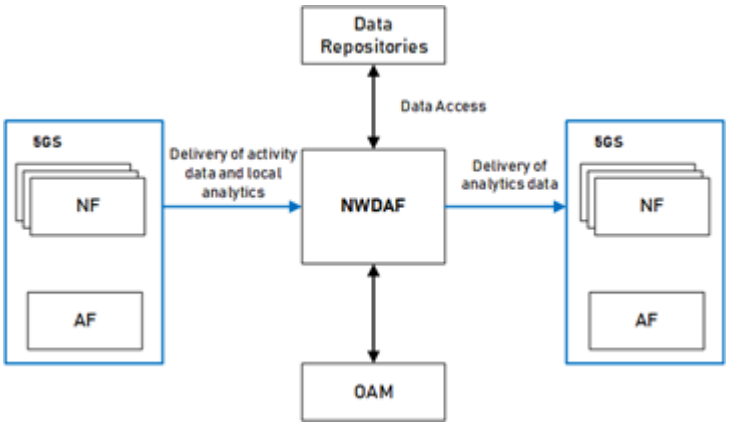


Figure 3 Distributed Architecture of NWDAF in R16 and R17

The Data Collection feature allows NWDAF to retrieve data from various sources (e.g., NFs such as AMF, SMF, PCF or other NFs) that include: OAM global NF data, behavioural data related to individual UEs or UE groups, metrics covering UE populations by spatial and temporal dimensions (e.g., per region for a given time period) For this purpose, NWDAF may use Generic Management Services (defined in TS 28.532) or Exposure Services provided by NFs/AFs to retrieve data not provided by OAM. In the case of network slices, NWDAF must determine which NF instance(s) of the relevant NF of a slice serve the UE or group of UEs (S-NSSAI(s) may assist in this determination).

Data collection procedures of NWDAF should allow data collection with the appropriate granularity. The Data Collection from NFs/AFs is based on AMF, SMF, UDM, PCF, NRF and AF services. This mechanism is used to obtain information on UEs. The information obtained from OAM may include NG RAN or 5GC performance and fault measurements as well as 5G end-to-end KPIs.

Data analytics performed by NWDAF are either statistical information about past events or predictive information. The NWDAF analytics include:

- Slice load level related network data analytics.
- Observed Service experience related network data analytics.
- NF load analytics.
- Network Performance Analytics.
- UE related analytics.
- User data congestion-related analytics.
- Data congestion-related analytics
- QoS Change analytics.

Zero-Touch Service Management (ZSM)

The ZSM framework, defined by ETSI, introduces closed-loop (CL)-based automation that includes an analytical function (AF) and a Data Collection Function (i.e., monitoring) that are interfaced with several internal as well as various external entities (authorized by ZSM). Figure 4 visualizes the closed loop and its functions within the ZSM framework.

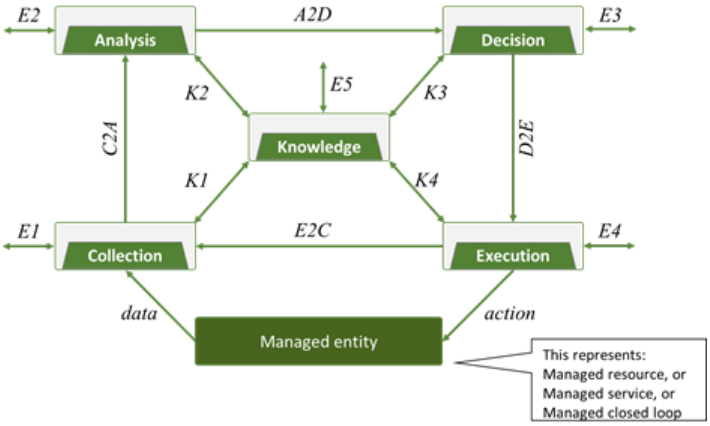


Figure 4 Functional View of a Closed Loop and its Functions within the ZSM Framework [35]

There are two functions closely related to MS and AE. The Collection Function is responsible for gathering and pre-processing data from managed entities (e.g., VNFs) or from external sources (e.g., context awareness positional data). Here the data can have different formats and be transferred from one or more sources (databases or streams) to a destination where it can be stored and further analysed. Since the data have different origins, each source must be transformed so that it can be analysed in conjunction with data from other sources.

The Analysis Function is responsible for deriving insights from available data gathered at the collection stage as well as historical data. An insight is jointly extracted from the data and the appropriate context. An example of an insight might be the conclusion that congestion has occurred in a set of resources, and the context might be the location, time and date, the service affected, the users involved, and the underlying set of slice resources. Insights can determine the root cause and locate it in the network. Deriving insights is therefore a continuous process that can be extended with new data. Analysis is able to continuously improve its results and provide better decision options to the decision function.

2.4 MonB5G Novelty and Contributions on MS/AE

The developed MS and AE are shifting their operations toward a decentralized, data-driven framework with little or no human intervention. To automatically manage a large number of slices, the developed MS and AE are embedded in the slices as distributed AI-based management entities that are locally deployed in different technical domains but interoperate together to automatically manage the corresponding slices focusing on service quality, energy efficiency, and communication resource optimization. We are mainly bringing in the following novel techniques here:

- The distributed MS entities, as part of the Slice Management Layer (SML), fulfil tight metrics sampling loops for the entities in the Slice Function Layer (SFL) where the management data/tasks are generated, so that the communication overhead introduced by MS itself is minimized.
- Additional MAPE-based embedded element managers (EEMs) are deployed with the functional entities to support fine-grained local telemetry collection. It also provides additional flexibility of specific aggregators for various AE and DE functions.
- The configurations of the distributed MS entities are defined and triggered by the AE/DE entities with AI-based policy-driven mechanisms that represents a decisive step towards a highly automated slice-level monitoring system.
- FL is extended for decentralized resource estimation to maintain low SLA violation. This AE feature introduces a set of well-designed statistical constraints for distributed network management with enhanced federated learning. The novel feature facilitates decentralized resource allocation in network slices while guaranteeing very low violation of SLA.
- Distributed neural networks are employed to move analysis operations locally. The distributed AE entity can support slices that are deployed in different domains. It facilitates prediction of sophisticated KPIs that depend on the performance of all components (VNFs, links, across domains) of a slice. The prediction offloading mechanism is data-drive, learn to optimize itself based on actual situations.
- We enhance AI-based traffic load prediction by improving the learning procedure with context-aware loss. Traffic load forecasting is essential for many downstream tasks such as resource allocation and admission control. The innovative AE feature can predict traffic load for any technological domains. This feature integrates additional regularizations to model penalties for over and under allocation of resources as well as resource reallocation settings. By ensuring that the right number of resources are provisioned to a network slice when needed, it significantly reduces the likelihood of SLA violations, and guarantees user perceived QoS.

- Local fault detection is enhanced with neighbourhood information. This AE feature identifies local anomalies based on learned normal behaviours of monitored entities. By employing a graph-based neural network, we integrate the information of neighbourhood entities. The anomalies can thus be better detected, as this feature considers not only the status of the monitored entity itself, but also a globe view of the related entities.
- We accomplish cross-domain anomaly detection with distributed optimization. All nodes in the network can obtain the global minimum value by communicating only with their neighbours, without the need of a central coordinator. The AE feature is asynchronous, namely, nodes can activate at any time without having to wait for any other specific event in the network. Moreover, this feature does not require data exchange among nodes, which largely reduces communication overhead.

2.5 Structure of the Deliverable

The main technical chapters of the deliverable are listed in Table 1. Here we also map the tasks defined in the grant agreement (GA) to the outputs reported in this deliverable to clarify and position the innovative contributions within the framework of the MonB5G project.

Table 1 Deliverable Structure and Mapping with Project Tasks

Chapter	Description	Task(s)	Starting Month
2	Introduce the motivations, state-of-the-art, as well as main contributions of the proposed MS/AE solutions	T3.1 - T3.3	M4
3	Present the major strategies and techniques explored by MonB5G MS/AE to achieve good scalability	T3.1 - T3.3	M4
4	Describe the 5G oriented datasets generated by the project, and the established simulators	T3.1	M4
5	Describe the position of MS/AE in the MonB5G architecture, and the interactions between MS/AE and other management components.	T3.1 - T3.3	M4
6	Report the design and implementation of the monitoring system, including MS at different technical domains, selected telemetries that are important for tracking the slice status, as well as the implementation details, visualization of some samples.	T3.1	M4
7	Introduce the MonB5G AE for KPI prediction, including local KPI prediction, cross-domain KPI prediction, as well as network aware KPI prediction	T3.2	M7

8	Report the AE for fault management, including local fault management, and outlier identification in networks with decentralized optimization	T3.3	M7

3 Scalability of MonB5G MS/AE

A key focus of the MonB5G project is to solve the scalability issue of network slice management, i.e., a scalable system to administrate a massive number of coexisting network slices with different technical requirements. As essential components of the platform, MS and AE are developed in a distributed and programmable framework. They distribute AI-enabled management operations at different levels of the management hierarchy and provide significant complexity reduction and fast analysis of the slices. Unlike the centralized management systems, the monitoring and analytics functionalities of the proposed management platform are embedded locally as part of the slices with the AI-driven enablers to achieve high scalability, especially for slices in multi-domains. To strengthen AE, novel distributed machine learning techniques, such as federated learning and distributed neural networks, are extended and adapted to fit the distributed management architecture. The resulting solutions effectively support the desired scalability.

In this chapter, we present the scalability of the MonB5G MS and AE. We start with the current challenges in scalable slice management and then discuss the vision and strategy of MonB5G, along with the technical contributions in MS and AE, to improve the scalability of slice management systems. Finally, we summarize the main achievements of the work package related to scalability.

3.1 Scalable AI-driven Network Management

With the deployment of virtualisation technologies such as Virtualized Network Functions (VNFs) and Software Defined Networks (SDNs), network slicing is being introduced as a key technical advancement to provide tailored communication services with different performance requirements, functionalities and time spans. In the future, 5G networks are expected to support massive number of network slices for a variety of vertical applications, which will benefit innovative business models, not only for Mobile Network Operators (MNOs), but also for MNO customers, who can dynamically request and negotiate services and performance to achieve personalisation, resource and cost efficiency. However, the high number of parallel, flexible slices brings additional challenges for network management, especially scalability.

Most existing network management systems, such as MANO, are problematic in managing numerous slices in a scalable manner, especially in the presence of multiple technical domains as often envisioned in the context of network slicing. A network slice consists of a set of interconnected VNFs, each of which encapsulates specific sub-services to perform functionalities for which it is designed. To meet the service requirements defined by the customer, a virtual subset of the physical resources of the network infrastructure are allocated to the slice. Depending on the exact definition of the service, the slice, i.e., a group of connected VNFs may be distributed across multiple network domains. Therefore, the slice management systems should perform administration tasks in e.g., RAN, Edge and Cloud, which is different from the traditional network management. The current MANO solutions mainly focus on centralised approaches. In this scheme, the communication between the orchestration entity and the distributed networking entities are intensive, as shown in Figure 5:

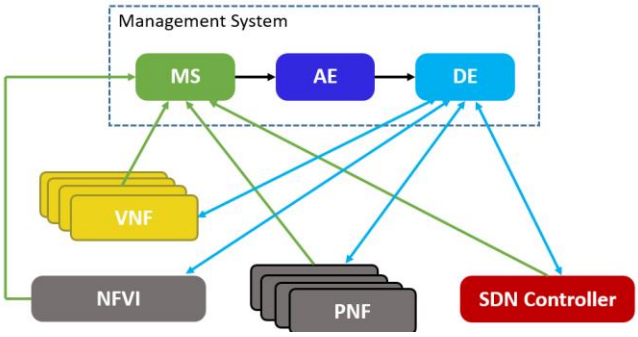


Figure 5 Centralized Network Slice Management

On the one hand, the monitoring information is forwarded to the centralized administration, on the other hand, the management decisions related to CHOP (Configuration, Healing, Optimization, and Protection) of each slice are returned to the distributed network components after analysing the data from the managed entities. Thus, the centralized management approaches cause significant problems:

- Large traffic overhead
- Unexpected delays due to delayed responses

When the number of slices increases, the situation becomes critical. The size of the monitoring data increases dramatically, especially when the slices span multiple domains. In each domain, the slices generate enormous monitoring telemetries. Transferring the management-related data to the administration centre consumes additional communication resources, and introduces unexpected latencies due to time cost of communication between the local network entities and the central administration. In domains (e.g., RAN) with stringent time constraints, this problem is critical. Overall, the limited scalability of centralized orchestration approaches is not applicable when managing a large number of slices. Considering the high variability, there are two main requirements for the novel slice management systems:

- Ensure up-to-date monitoring information with low overhead during the administration process.
- Enable automatic online analysis and decision making in response to unexpected network dynamics.

Thus, an efficient and intelligent closed-loop solution of AI driven monitoring, analysis, and actionable decision making must be to orchestrate a massive number of parallel network slices. However, as implied by the Universal Scalability Law (USL) [36], increasing processing resources does not guarantee scalability, as it also increases the complexity of their management, the degree of contention in the system and suffers from a lack of collaboration among distributed decision entities. From a technical point of view, as shown in Figure 6, a scalable architecture must make a trade-off between the following factors:

- Using shared resources to minimise contention but also avoiding complex management of unnecessary resources;
- Information flow through the exchange of only compressed parameters instead of raw data;
- Degree of collaboration by enabling the exchange of inferences between decentralised analytics/decision engines, avoiding that they end up in competing approaches or overly collaborative situations.

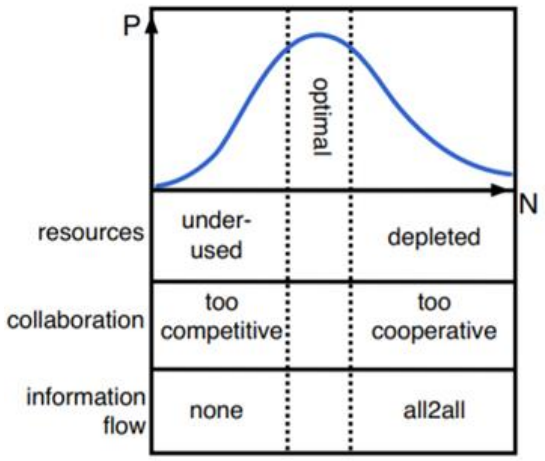


Figure 6 Scalability trade-offs [37]

MonB5G aims for at least *linear scalability*, i.e. the network is able to handle a high number of concurrent slices without sacrificing lifecycle management (LCM) KPIs or AI performance. From a design viewpoint, the MonB5G solution fulfils the above trade-offs because it has the following features:

- **Hierarchical architecture per technological domain** [38]: Intuitively, in flat architectures, a considerable increase of admitted slices would lead to performance degradation due to a large number of peer-to-peer message flows or AI-based inference exchanges between the different distributed local management entities. Similarly, in purely centralized architectures, a large amount of raw monitoring data originating from a large number of parallel slices must pass through multiple technological domains to be analysed and used for centralized decision-making. This poses a challenge to transport and processing queues, leading to contention and large delays/overhead. In contrast, MonB5G defines a fine-grained, distributed, yet hierarchical slice management architecture, where analysis and decision are located close to the monitored resources. At the same time, for a higher escalation level (slice-level, domain-level or IDMO-level) is considered, where a specific LCM coordination, inference aggregation, or long-range policy takes the lead, based only on compressed parameters sent by local entities.
- **MS/AE/DE defined per slice:** This means fewer contention compared to centralised approaches.
- **Decentralised and collaborative analysis/decision making:** The scope of collaboration is limited to a well-selected subset of AEs/DEs for efficiency reasons. This implies a per slice-type or per-domain selection policy or selection of AEs that provide a better performance improvement.
- **Compressed models sharing:** Trade-off between sharing raw data and not sharing data. In this case, only compressed models or data are shared across the network.

3.2 Vision and Strategies of MonB5G for Scalable MS and AE

As essential components of the MonB5G platform, MS and AE embed into the architecture of the distributed management system (please refer to D2.4 for details) and enable compelling performance in terms of scalability with mainly two novel contributions:

- Distribution of hierarchical monitoring and analysis operations in different domains of network slices.
- Tailoring and extending novel distributed AI technologies for efficient and accurate analysis.

The hierarchical orchestrator has recently gained attention due to its flexibility in distributing a variety of management tasks across entities targeting different network domains. In particular, a network slice can be viewed as a combination of a set of sub-slices, each of which is often deployed in different domains. To overcome centralised approaches, which are problematic due to traffic overhead and delay, multiple management elements, such as monitoring and analysis, are introduced into each domain-specific sub-slice, which is logically closer to the pool of resources to be managed, enabling faster and even proactive slice KPI prediction and fault detection. The hierarchical orchestration provides different levels of centralisation for monitoring and analysis. The hierarchical structure allows these management tasks to be performed at lower levels, effectively limiting monitoring overhead and reducing the reaction time.

Furthermore, we see distributed AI as the key to automated and scalable management for optimal distribution of analysis tasks across different hierarchical layers. On the one hand, the AI functions directly analyse the status of network entities without human intervention so that the requested KPIs can be efficiently predicted on pre-defined time scales (often stringent) on RAN, transport, NFV infrastructures, and E2E network slices. Thus, the AI-based analytics functions provide latency improvement, and enable processing of more network slices in a given time. On the other hand, deploying AI methods in a distributed architecture is not straightforward. In such a decentralized management framework, the analysis function is expected to be executed locally, i.e., near the data over which it is generated to reduce traffic load and enable fast predictions. However, the local data is not capable of providing an overall view of the network status. The accuracy of local predictions needs to be calibrated in an effective way.

In addition, local resources are usually limited, so the capabilities and flexibility of local AI models are less powerful to handle complicated situations that often occur in the context of a highly dynamic network environment. Therefore, the novel distributed AI technologies should be leveraged and extended to optimize the distributed analysis operations. In particular, the local AI models need to integrate the information and patterns learned with the global data. When the local AI-driven analysis is unable to solve the newly arrived complicated tasks, it performs initial inference task and provides the upper layers with the refined information (instead of the raw monitoring data) to facilitate problem solving and reduce the communication overhead. In this context, the policy of information exchange between adjacent levels (horizontal and vertical) is particularly critical, as it affects the overall ability of AE to provide the requested analysis functions promptly. Therefore, the distributed AI technologies are adapted to learn the best distribution of analysis tasks in the orchestrator hierarchy according to the current network status.

3.2.1 SCALABLE MS

The MonB5G monitoring system is designed to collect detailed information about the current status of network slices, which are often deployed in multiple domains. To provide up-to-date monitoring for online analysis of slice KPI and reconfiguration in response to unexpected network dynamics, scalability of MS is an

essential aspect for managing a large number of slices. To achieve the goal of scalability, we employed several strategies in the design and implementation of MS, which are discussed in this section.

The *first strategy* that facilitates the scalability of MS is its distributed architecture. The MS is designed based on the microservice architecture and implemented as a cloud-native application in a Kubernetes cluster. This means first that the MS benefits from the auto-scaling features of Kubernetes. More specifically, the resources of MS can be adjusted depending on the load; this requires defining the auto-scaling rules in the Kubernetes deployment. Kubernetes, as the orchestrator of the application, monitors load of the system and adjusts the number of pods based on the auto-scaling rules to keep the system load below the given threshold. Second, in MS, all components, including the Sampling Functions (SFs), are implemented as containers and deployed as pods in the Kubernetes cluster. On the other hand, each cluster can have multiple working nodes which means that the system can be scaled by increasing the number of slices to be monitored and consequently the number of SFs by adding more working nodes.

Figure 7 shows the distributed deployment of MS across three physical nodes. In this deployment, the messaging bus, implemented by Kafka, is *stretched* in all three nodes (this "stretched cluster" feature is supported by Kafka). This means that any message published by any producer on any node is available to all consumers on all nodes. In this deployment, Node 1 only hosts the manager and the TSDB (Time Series Data Base) pods. Node 2 and Node 3 host three different sampling functions that run separately. All of these components communicate over the Kafka bus. If more sampling functions are needed and the resources on Node 2 and Node 3 are not sufficient, another node can be added to the cluster. This deployment demonstrates the *linear scalability* of the MS.

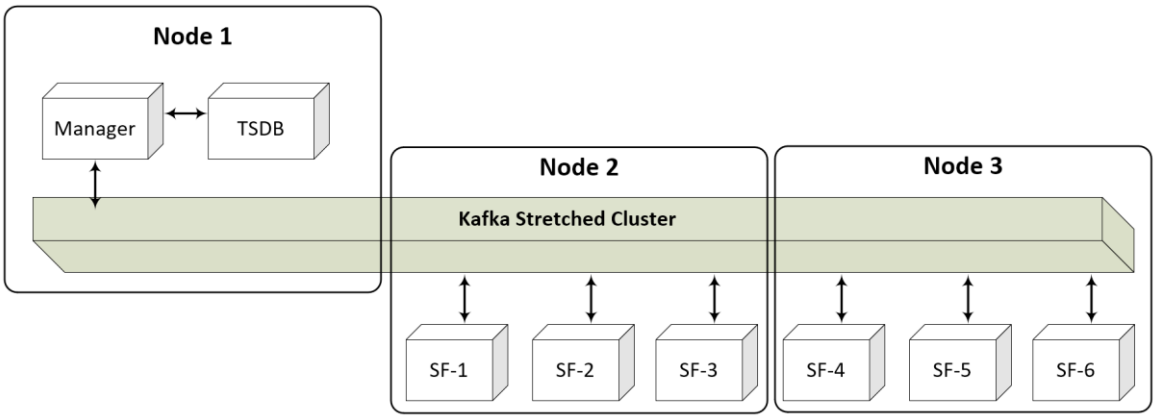


Figure 7 Distributed deployment of the monitoring system

The *second strategy* to achieve the scalability goal is the hierarchical architecture of the monitoring system, i.e., it is possible to have multiple sub-monitoring systems which are collaborating in a master-slave pattern, and the data monitored by the slave MS can be further gathered by the master MS as needed. With this hierarchical structure, monitoring tasks can be performed at lower levels, limiting data traffic and reducing the reaction time. Monitoring information can be extracted directly from distributed MS entities and aggregated locally. Figure 8 shows the hierarchical deployment of the monitoring system. It is a hierarchical deployment where the monitoring systems in the lower part of the figure are the low-level ones that directly monitor the resources such as VNF and PNF. However, the MS in the upper part is the higher-level MS, which

does not directly monitor the resources. Instead, it connects to the broker or TSDB of lower level MSs to gather data from them. When using MS in a multi-domain network, a low-level MS can be instantiated for each technological domain, and the high-level MS would be a centralized MS that gathers data from multiple domains.

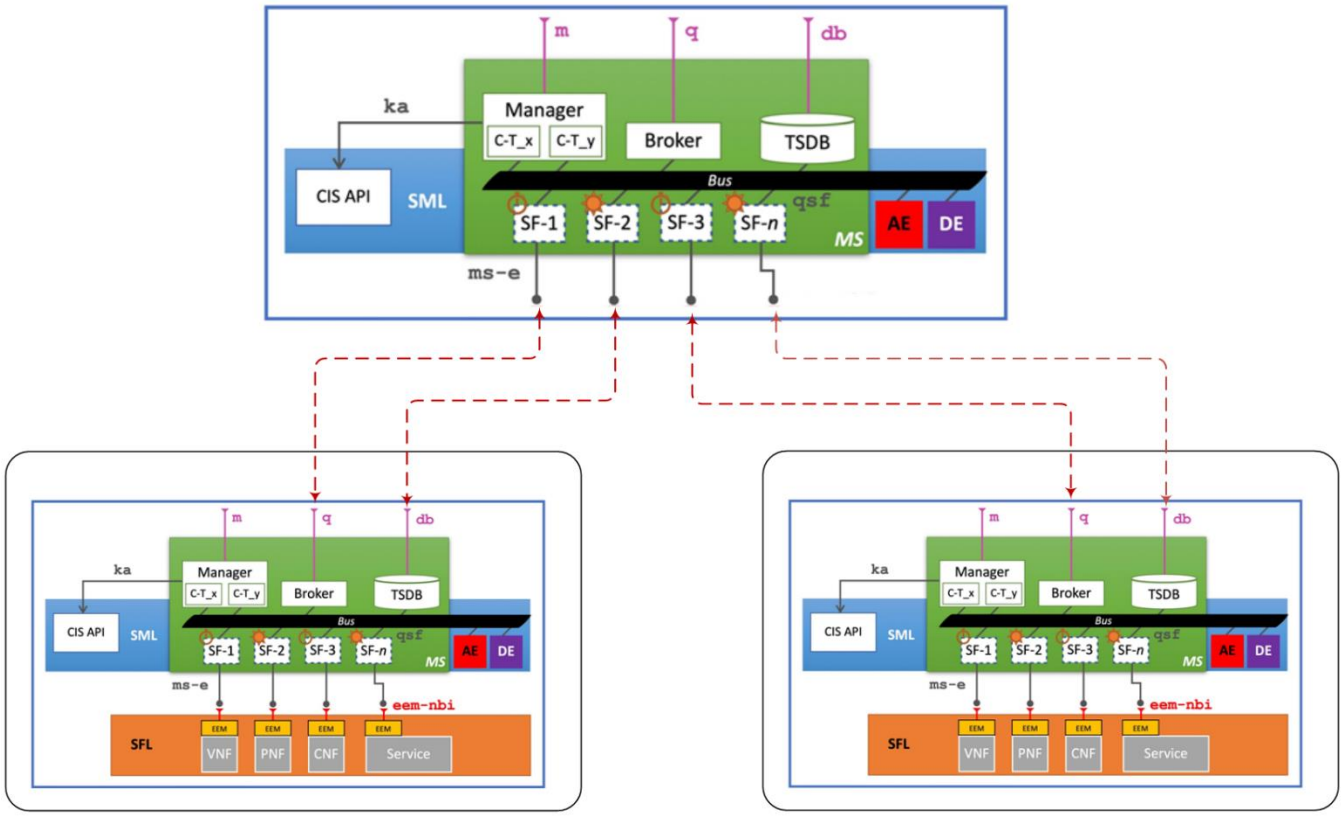


Figure 8 Hierarchal deployment of monitoring systems

In this figure, the monitoring systems at the bottom of the figure primarily monitor their corresponding domains. The telemetry data gathered by these monitoring systems is available through the "q" and "db" interfaces. The sampling functions in the central MS (the monitoring system in the upper part of the figure) can use these interfaces to gather the data from the other MS. An important point to emphasize is that the sampling functions in the central MS do **not** need to gather all the raw data sampled by the MSs of domains I, Instead, many different preprocessing methods such as filtering, aggregation, and even AI-based compression/extraction are applied to the local data before transmission, largely reducing the amount of data for the central system.

The *third strategy* to achieve the goal of scalability is to reduce the footprint of the system. To reduce the resource footprint of the whole Slice Management Layer (SML), MS components are shared among other administrative components i.e., AE and DE. This can be partially seen in Figure 8, where the Streaming Bus also reaches AE and DE. Moreover, the TSDB is also accessible to AE and DE (via the ***db*** reference point, as shown in Figure 8) so, they can use it to store analytics or policies, respectively. Additionally, Sampling Functions can allow more than one *TARGET* for the same *eem-nbi*. This means that a single sampling function

can sample multiple telemetry data if the corresponding EEM supports it. This feature can significantly reduce traffic between MS and EEMs and improve the scalability of the system.

The *final strategy* for improving the scalability of the monitoring is the way sampling functions are implemented. Each sampling loop is implemented by a single Docker image. In this way, the loop resides inside the Docker image, and there is no need to frequently create and destroy the sampling function container. More specifically, the container of the sampling function is deployed only once and receives samples from the EEM on a regular basis.

3.2.2 *SCALABLE AE*

The AE is designed to analyze the status of network slices using telemetry data collected by MS. Its outputs, i.e., the analysis results, are then reported to DE as key indicators to learn from and infer actionable decisions to maintain and optimize the slice performance defined in SLAs. Since slices can be deployed in multiple infrastructure domains and the slice-level KPIs are diverse, there are a variety of analysis operations performed in AE, such as traffic load forecasting for RAN, resource usage prediction for Edge, and SLA violation detection for Cloud, which are computationally complicated and may potentially involve data from multiple domains.

As the number of slices increases, the scalability of AE becomes critical. To address the problem, we first embed AE into the distributed MonB5G platform to achieve good scalability due to its hierarchical and decentralized architecture. More importantly, AE extends the novel distributed AI technologies to optimally distribute the analysis operations across different layers and entities. Specifically, we identify the following strategies to extend the distributed AI methods to improve scalability:

1. **Analysis operations should be distributed among different layers of the management hierarchy.** In this way, analysis will make full use of local computing resources and provide online analytics services to function entities without unexpected delay.
2. **The analysis should be executed close to the location where the data is generated.** This will significantly reduce communication overhead and time costs.
3. **The distribution of analysis operations among management entities should be optimized by AI-driven methods** that depend on the current and (forecasted) future network status. Although a threshold based static solution can be a good starting point, novel distributed AI, such as federated learning and distributed neural networks, will be key to optimizing analysis operations to address the challenge of high network status variability. In addition, these AI techniques facilitate the local AE to integrate a global view of slice status during local analysis.

Distributed AI technologies play a critical role in improving scalability of the AE. Although AI methods have been widely integrated into network management, such as the 3GPP defined Network Data Analytics Function (NWDAF) in the 5G Core network and ETSI-defined zero-touch service management (ZSM) for management automation, most existing AI-driven solutions focus on centralized solutions and the use of AI methods under a distributed management platform is still challenging in terms of the distribution of the algorithms themselves. For distribution of analysis operations between adjacent layers, and between multiple management entities on the same layer, the AE extends the following novel AI technologies:

- **Federated learning (FL).** The FL methods have been viewed as a key enabler for distributed management. They make no assumptions about the local data, and therefore allow for heterogeneous local data with highly imbalanced data size, which meets the requirements of a highly dynamic network status. The FL methods enable network scalability by distributing and performing most of processing locally and rapidly. Figure 9 illustrates how FL is used to perform local inference, while still integrating the comprehensive information via a global model.
- **Distributed ML**. These techniques, e.g., distributed neural networks, are a set of vital tools in scaling network management. A typical scenario of distributed ML is to move the training and inference mechanism from centralised architectures of Core/Cloud to Edge which are closer to the users. Thus, the latency introduced by data communication and execution of ML models at the Core/Cloud level can be significantly improved. Figure 10 illustrates the key ideas of the proposed AE with distributed NNs. It distributes operations between local AE entities and high-level entities by optimizing the predictive confidence, where local models are the bottom layers of a deep NN.
- **Representation learning** with e.g., Deep Neural Networks (DNNs). DNNs are typically used to learn data representations of unstructured data. They are now used to compress the management data to reduce the communication overhead. Meanwhile, the learned low-dimensional representations still preserve the intrinsic properties of the data, which then guarantees the analytics quality, e.g., predicting the network/VNF status and load. Replacing LSTM with GRU layers in the local AE significantly reduces training and testing times and achieves a set of results comparable to the state-of-the-art. Additionally, incorporating Convolutional Neural Network based learning is a more efficient method for extracting the local patterns required for scalable neural networks.

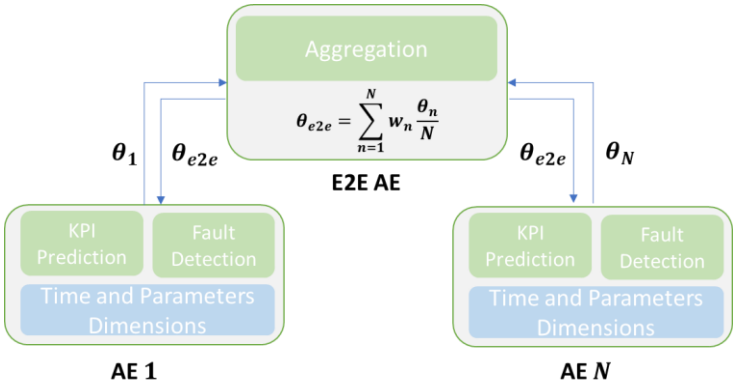


Figure 9 Decentralized AE with federating learning

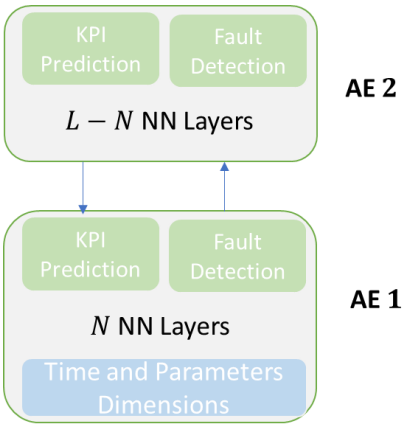


Figure 10 Decentralized AE with distributed neural networks, where the local models are the bottom layers of the NN.

The MonB5G AE is embedded in the decentralized hierarchical management architecture (defined in [D2.4]) and adapts the novel distributed AI techniques to ensure high quality of online analysis functions and further reduce communication cost, computational complexity and transmission delay. By automatically distributing the analysis operations to the different layers and entities with AI-driven policies, AE achieves good scalability to manage a massive number of parallel slides without requiring intensive communication between central administrative entities and networking ones.

3.3 Major Achievements of MonB5G on Scalability

To show scalability, MonB5G relies on the trend of some KPIs compared to the number of admitted slices and supported slices compared to network resources. These trend models will be valid and extrapolated directly to the massive slicing regime. Several evaluation mechanisms will be considered for this purpose:

- Characterize for example slice setup time and show that it exhibits a linear/sublinear trend compared to the super-linear behaviour of state-of-the-art centralized Management and Network Orchestration (MANO) solutions
- Demonstrate that overall AI performance (e.g., accuracy, loss) is not degraded while guaranteeing that induced overhead does not increase super-linearly (e.g., exponentially) with the number of slices as a result of MonB5G small footprint decentralised AE/DE architectures and algorithms.
- Ensure that the trend in the number of slices supported with fulfilled SLA scales at least linearly with the increase in allocated resources, compared to centralized MANO which might exhibit sublinear scalability.

4 5G Data and Simulation

Data plays an important role in the development of AE. The proposed decentralized management plane will use data-driven mechanisms based on AI algorithms. Training and testing these algorithms require large data sets. However, it is difficult to acquire real network data from operators, due to concerns of data privacy of end users and commercial confidentiality. To address the problem, we construct two 5G simulators (with different focuses), and collect synthetic datasets to develop the proposed analysis functions. Part of the datasets will be published as a result of the MonB5G project for the benefit of the 5G/B5G research community.

4.1 Publicly Available Data Related to 5G

In section 3.3 of deliverable D3.1 [3] (previous version of this deliverable) we analysed an initial list of public datasets that we identified, a subset of which we have already leveraged (for example, the Milan Dataset). However, most of these datasets covered 3G/4G network traffic, while our primary interest is in datasets generated by 5G networks.

To find public data related to 5G, we used search engines that focus on datasets, such as Google Dataset Search[14] & Kaggle Datasets[15]. To narrow our search to datasets useful for research purposes, we also leveraged IEEEDataPort[16], ResearchGate[17] and reviewed project outcomes published as datasets on Zenodo[18] and OpenAIRE Explore[19]. The following is a list of public 5G datasets.

Dataset: Large-Scale Dataset for the Analysis of Outdoor-to-Indoor Propagation for 5G Mid-Band Operational Networks

Found via: ResearchGate & OpenAIRE/Zenodo

Published: February 5, 2022

Source (Zenodo): https://doi.org/10.5281/zenodo.5975814

Description: Dataset that supports publication with the same title[20]. The dataset includes measurements of channel power delay profiles from two commercial 5G networks in Band n78, i.e., 3.3–3.8 GHz. Such measurements were collected at multiple locations in a large office building in the city of Rome, Italy by using the Rohde & Schwarz (R&S) TSMA6 network scanner during several weeks in 2020 and 2021. A primary goal of the dataset is to provide an opportunity for researchers to investigate a large set of 5G channel

[14] https://datasetsearch.research.google.com

[15] https://www.kaggle.com/datasets

[16] https://ieee-dataport.org/

[17] https://www.researchgate.net/

[18] https://zenodo.org/

[19] https://explore.openaire.eu/

[20] https://doi.org/10.3390/data7030034

measurements, aiming at analyzing the corresponding propagation characteristics toward the definition and refinement of empirical channel propagation models.

Applicability: No relevance to MonB5G use cases.

Dataset: Selected processed 5G base station RF-EMF measurement data

Found via: Google Dataset Search & OpenAIRE/Zenodo

Published: January 28, 2022

Source (Zenodo): https://doi.org/10.5281/zenodo.5914224

Description: This dataset presents selected processed 5G base station radio frequency electromagnetic field (RF-EMF) measurement data acquired under measurement campaign in outdoor environment carried out at 5G-VINNI UK facility, based at British Telecom (BT)'s Adastral Park in Ipswich, UK. This dataset supports the findings presented in Section 5 of the deliverable report D1 of the H2020/EMPIR 5GRFEX project entitled: "Metrology for RF exposure from Massive MIMO 5G base station: Impact on 5G network deployment".

Applicability: Type of measurements not of interest.

Dataset: 5G deployment dataset

Found via: OpenAIRE

Published: January 6, 2022

Source: https://data.mendeley.com/datasets/s2f3xvgnrr/2 (Mendeley Data DOI: 10.17632/s2f3xvgnrr.2)

Description: Dataset that supports the publication "5G network deployment and the associated energy consumption in the UK: A complex systems' exploration"[21], which took the UK as an example to investigate the spatiotemporal dynamic characteristics of 5G evolution, and further analysed the energy consumption and carbon footprint of 5G networks, as well as the consequent change in the operating expenses pattern. The input dataset of the study covers postcode, area, number of mobile users, mobile user density, and number of base stations and base station density of the specific country.

Applicability: Dataset type not of interest.

Dataset: 5G Campus Networks: Measurement Traces

Found via: Google Dataset Search & IEEE Dataport

Published: December 15, 2021

Source (IEEE Dataport): https://ieee-dataport.org/open-access/5g-campus-networks-measurement-traces (DOI: https://doi.org/10.21227/xe3c-e968)

Source (GitHub): https://github.com/justus-comnets/5g-campus-measurements

Description: Dataset supporting the publication "5G Campus Networks: A First Measurement Study"[22]. Contains packet captures (PCAPs) for 5G SA and 5G NSA.

Applicability: Could be used for dataset generation.

[21] https://doi.org/10.1016/j.techfore.2022.121672

[22] https://doi.org/10.1109/ACCESS.2021.3108423

Dataset: 5G: Energy Efficacious Methodology Dataset

Found via: OpenAIRE

Published: October 8, 2021

Source: https://data.mendeley.com/datasets/5z37x49p74/1 (Mendeley Data DOI: 10.17632/5z37x49p74.1)

Description: The data consists of power consumption of 5G BSs, various demographic areas and LTE cellular mobile DRX cycles. Researchers can use this data in the finding of energy consumption in 5G networks. It can also help to carry out study for the energy-efficient 5G system model. The dataset can be utilized for minimization of 5G BS power consumption, long cell phone battery life and energy-efficient system model.

Applicability: Only high-level parameters are included.

Dataset: A variegated look at 5G in the wild: performance, power, and QoE implications

Found via: Citations

Published: August 9, 2021

Source (GitHub): https://github.com/SIGCOMM21-5G/artifact

Description: Dataset that supports publication with same title[23]. The authors used 5G Tracker[24] (app available under license by UMN, see below) to capture the measurements.

Applicability: Could be used for dataset generation.

Dataset: 5Gophers v1.0 (Commercial 5G Network Performance)

Found via: Google Dataset Search & IEEE Dataport

Published: November 2, 2021

Source (IEEE Dataport): https://ieee-dataport.org/open-access/5gophers-v10-commercial-5g-network-performance (DOI: https://dx.doi.org/10.1145/3366423.3380169)

Source: https://fivegophers.umn.edu/www20

Description: Dataset supporting publication "A First Look at Commercial 5G Performance on Smartphones"[25], based on conducting a first measurement study of commercial 5G performance on smartphones by closely examining 5G networks of three carriers (two mmWave carriers, one mid-band 5G carrier) in three U.S. cities. The authors conducted extensive field tests on 5G performance in diverse urban environments, systematically analyzed the handoff mechanisms in 5G and their impact on network performance, and explored the feasibility of using location and possibly other environmental information to predict the network performance. The authors also studied app performance (web browsing, HTTP download, and volumetric video streaming) over 5G. The same authors also described the 5G Track (available under license by UMN)

[23] https://doi.org/10.1145/3452296.3472923

[24] https://license.umn.edu/product/5g-tracker-android-application-for-collecting-and-visualizing-5g-performance-data

[25] https://doi.org/10.1145/3366423.3380169)

they developed for their study, in publication "5G tracker: a crowdsourced platform to enable research using commercial 5g services"[26].

Applicability: Could be used for dataset generation. 5G Tracker app could also be used for data capturing.

Dataset: Lumos5G: Mapping and Predicting Commercial mmWave 5G Throughput

Found via: Citations & IEEE Dataport

Published: November 2, 2021

Source (IEEE Dataport): https://ieee-dataport.org/open-access/lumos5g-dataset
(DOI: https://dx.doi.org/10.1145/3419394.3423629)

Description: Dataset supporting publication of same title[27], where the authors conducted a measurement study of commercial mmWave 5G services in a major U.S. city, focusing on the throughput as perceived by applications running on user equipment (UE). Through extensive experiments and statistical analysis, they identified key UE-side factors that affect 5G performance and quantified to what extent the 5G throughput can be predicted. The authors then proposed a composable machine learning (ML) framework that judiciously considered features and their combinations and applied state-of-the-art ML techniques for making context-aware 5G throughput predictions.

Applicability: Could be used for dataset generation.

Dataset: A series of 5G measurement tools and dataset

Found via: Citations

Published: July 30, 2020

Source (GitHub): https://github.com/piaobozaizai/5G_measurement

Description: Tools and dataset supporting publication "Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption"[28]. They authors demystify operational 5G networks through a first-of-its-kind cross-layer measurement study. Their measurement focuses on four major perspectives: (i) Physical layer signal quality, coverage and hand-off performance; (ii) End-to-end throughput and latency; (iii) Quality of experience of 5G's niche applications (e.g., 4K/5.7K panoramic video telephony); (iv) Energy consumption on smartphones.

Applicability: Could be used for dataset generation, but more recent datasets above are preferable.

Dataset: Synthetic Data Set for Network Data Analytics Function (NWDAF)

Found via: Citations

Published: July 17, 2020

Source (GitHub): https://github.com/sevgicansalih/nwdaf_data

[26] https://doi.org/10.1145/3405837.3411394

[27] https://doi.org/10.1145/3419394.3423629

[28] https://doi.org/10.1145/3387514.3405882

Description: Synthetically generated dataset that supports publication "Intelligent network data analytics function in 5G cellular networks using machine learning"[29].

Applicability: Could follow similar approaches for generating datasets.

Dataset: Beyond throughput, the next Generation: a 5G dataset with channel and context metrics

Found via: ResearchGate

Published: June 2020

Source (GitHub): https://github.com/uccmisl/5Gdataset

Description: Dataset supporting publication with same title[30]. In this work, the authors presented a 5G trace dataset collected from a major Irish mobile operator. The first such publicly available dataset was generated from two mobility patterns (static and car), and across two application patterns (video streaming and file download). The dataset was composed of client-side cellular key performance indicators (KPIs) comprised of channel-related metrics, context-related metrics, cell-related metrics and throughput information. These metrics were generated from a well-known non-rooted Android network monitoring application, G-NetTrack Pro.

Applicability: Could be used for dataset generation, but more recent datasets above are preferable.

Dataset: DeepSlice & Secure5G - 5G & LTE Wireless Dataset

Found via: Google Dataset Search & Kaggle Datasets

Published: November 2, 2019

Source (GitHub): https://github.com/adtmv7/DeepSlice

Source (Kaggle): https://www.kaggle.com/datasets/anuragthantharate/deepslice

Source (CRAWDAD): https://crawdad.org/umkc/networkslicing5g/2022-03-22/

Description: Dataset that describes a Deep Learning model for 5G and Network Slicing (eMBB, URLLC, IoT) supporting two different publications:

- DeepSlice: A Deep Learning Approach towards an Efficient and Reliable Network Slicing in 5G Networks[31]
- Secure5G: A Deep Learning Framework Towards a Secure Network Slicing in 5G and Beyond[32]

Applicability: Dataset type not useful for MonB5G purposes.

4.2 Methodology of Data Generation with MonB5G Platform

We have simulated two synthetic 5G data for development of AI algorithms in AE, DE and security components. This section introduces design and architectures of the two simulators.

[29] https://doi.org/10.1109/JCN.2020.000019

[30] https://dx.doi.org/10.1145/3339825.3394938

[31] https://doi.org/10.1109/UEMCON47517.2019.8993066

[32] https://doi.org/10.1109/CCWC47524.2020.9031158

4.2.1 IQU DATASET

The synthetic dataset generation is handled by IQU's *in-situ* simulator called *VNFNet*. It is capable of generating synthetic datasets of 5G/B5G networks with real SDN-NFV enabled topologies. It was developed with migration to a production environment in mind and can be integrated with various RL algorithms via the OpenAI environment. Simulations are based on existing real topologies from *The Topology Zoo* datasets [39], and parameters are based on *in-situ* 5G measurements for URLLC use-case scenarios [40].

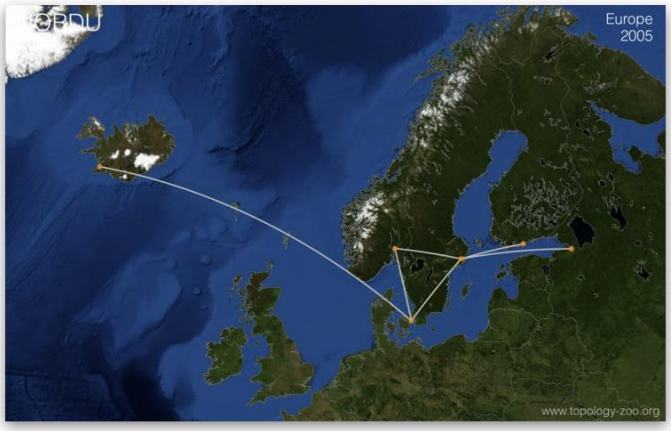


Figure 11 Real topology integration example. Nordu 2005 network from The Topology Zoo dataset [39].

The simulator has been validated in multiple publications, and its outs are supported by the works [40] [41]. The *in-situ* dataset generator was developed based on the review recommendations and the privacy concerns of the operators.

Simulator design

The virtualized network was simulated using Python. The network simulation is based on a fork of *Containernet* [42], an advanced branch of *Mininet* [43] a network emulator widely used in the literature, such as [44] and [45]. It simulates a realistic virtual network, VM hosting, switching, and application code for development and experimentation with SDN-NFV networks. The network topologies are taken from *The Internet Topology Zoo* dataset and were customized to meet the requirements of the use-case scenario. In addition, a custom *OpenAI* [46] Gym environment was developed to allow easy integration with RL and distributed RL solutions.

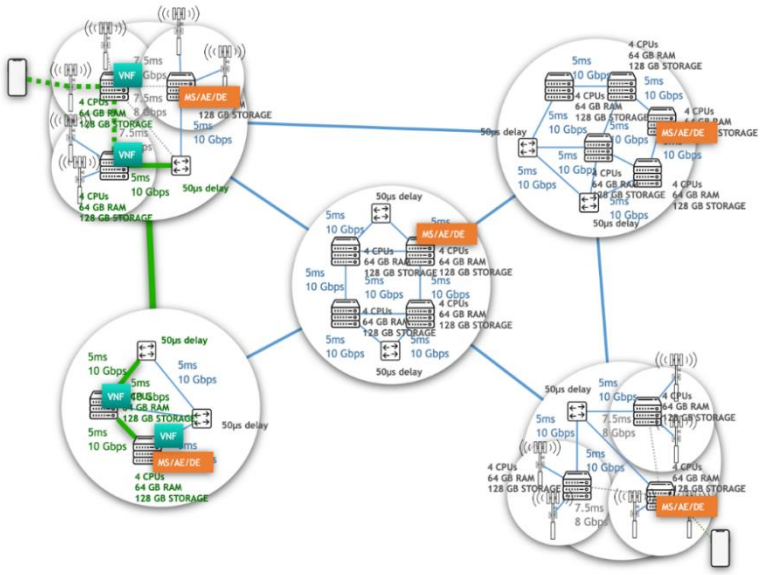


Figure 12 Simulated network graph example

Shown as Figure 12, network simulation is abstracted by graphs using the well-known NetworkX Python module, and the SDN routing is handled by Dijkstra's algorithm. Docker containers managed by Kubernetes are used to emulate VNFs and computational resource utilization is simulated using a Python script. The rest of performance emulation is based on the influential *Containernet* [42] project. The simulation is based on real networks and use-case scenarios such as user mobility or congestion can be simulated.

4.2.2 *EUR DATASET*

The dataset, identified by the name "**Eurecom AMF Resource Consumption Dataset**" (EARCD), was created with the help of the EURECOM 5G facility. An updated 5G UE emulator is used, which sends a large number of UE Attach Request messages in parallel. The 5G CN consists of elements based on OpenAirInterface (OAI).

The project uses this dataset to build ML models that run in the closed-loop management system. For instance, it is used in WP5 to detect mMTC attacks on the AMF. The dataset may be useful to researchers interested in 5G data and OAI AMF performance. It is made publicly available so that it can be easily reused by external parties to the project. The dataset is defined in D3.2 and is now accessible to only to project partners. It will be released to the public before the end of the project.

Simulator design

Regarding the UE, we used and updated a 5G UE emulator, we leveraged the '**my5G-RANTester**" with a script that allows emulation of real UE traffic (control plane) to communicate with 5G Core Network components. We used the emulator to simulate an event with different SUPI values selected. The traffic generated is similar to what real UEs would generate. My5G-RANTester follows the 3GPP Release 15 standard for NG-RAN (Next

Generation-Radio Access Network). With my5G-RANTester, it is possible to generate different workloads and test different functionalities of a 5G core, including compliance with 3GPP standards.

We used my5G-RANTester to send a large number of UE Attach Request messages in parallel to simulate traffic. My5G-RANTester is a tool to emulate control and data planes of the UE (user equipment) and the gNB (5G base station). Scalability is also an important feature of the my5GRANTester, which can mimic the behavior of a large number of UEs and gNBs accessing a 5G core simultaneously. Currently, the wireless channel is not implemented in the tool. The AMF and 5G CN components are based on OpenAirInterface (OAI). Additionally, we have developed an Element to dynamically allocate different resources (different RAM, CPU) and collect the actual resources consumed by the AMF. This collected data is stored in a CSV file.

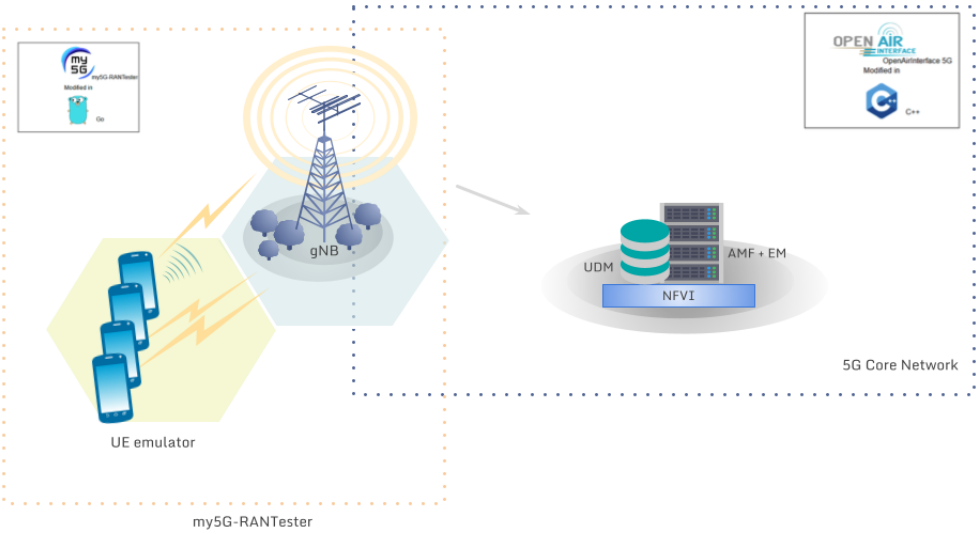


Figure 13 Framework used for the dataset generation

4.3 MonB5G Data for AI-Driven Network Management Research

Now let introduce the generated datasets, such as statistics of the datasets, meaning and range of attributes, visualization of some data examples.

4.3.1 IQU DATASET

Realistic synthetic data review and analysis

The generated data is stored in a secure file with JSON structure (shown in Figure 14) and generated per slice to be used by intra-slice multi-domain VNF management and orchestration DE. The state of all network elements can be normalized to enable AI-based decision-making. An additional file is generated to map the network state column values to the respective network element (shown in Figure 15).

```
297 }{
298     "Iteration": "38",
299     "State": "[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1, 1.0, 1, 1.0, 1, 0.25, 0.0625, 0.002]
        , [0.25, 0.0625, 0.002, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1, 1.0, 1, 1.0, 1, 0.25, 0.0625, 0.002],
        [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1, 1.0, 1, 1.0, 1, 0.25, 0.0625, 0.002], [0.0, 0.
        0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1, 1.0, 1, 1.0, 1, 0.25, 0.0625, 0.002], [0.0, 0.0, 0.0, 0.
        0625, 0.03125, 0.001, 0.125, 0.0625, 0.01, 1.0, 1, 1.0, 1, 1.0, 1, 0.25, 0.0625, 0.002], [0.0, 0.0, 0.
        0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1, 1.0, 1, 1.0, 1, 0.25, 0.0625, 0.002]]",
300     "ActionServer": "2",
301     "ActionDomain": "4",
302     "Reward": "41",
303     "Latency": "12.75",
304     "Rejections": "-1"
305 }{
```


Figure 14 Data Sample file


```
1  {
2      "6": [
3          "0-cpu-un",
4          "0-ram-un",
5          "0-hhd-un",
6          "1-cpu-un",
7          "1-ram-un",
8          "1-hhd-un",
9          "2-cpu-un",
10         "2-ram-un",
11         "2-hhd-un",
12         "3-bw-un",
13         "3-lat",
14         "4-bw-un",
15         "4-lat",
16         "5-bw-un",
17         "5-lat",
18         "56-s-cpu-n",
19         "56-s-ram-n",
20         "56-s-hdd-n"
21     ],
22     "13": [
23         "7-cpu-un",
24         "7-ram-un",
25         "7-hhd-un",
26         "8-cpu-un",
27         "8-ram-un",
28         "8-hhd-un",
29         "9-cpu-un",
30         "9-ram-un",
31         "9-hhd-un",
32         "10-bw-un",
33         "10-lat",
34         "11-bw-un",
35         "11-lat",
36         "12-bw-un",
```


Figure 15 Data Mapping file

These files are created with the intent of being used as a dataset for further analysis or exploitation by the Analytics Engine. The structure of the Data File can be analyzed as shown in Table 2:

Table 2 Attributes of the IQU dataset

Label	Type	Description
Iteration	unsigned int	Discrete runtime time reference
State*	2D array	Domain state space statistics*
ActionServer	unsigned int	ID of the intra-domain server that will receive the VNF in auction (Winner)
ActionDomain	unsigned int	ID of the domain that will receive the VNF in auction (Winner)
Reward	double	Shared RL agent slice-wide reward
Latency	double	Average service latency measured after applying the new VNF placement
Rejections	unsigned int	Number of rejected users due to SLA violation

The Mapping File is an automatically generated JSON file that links the items of the State* 2D array of the Data Sample files to the network elements and their function. These files are generated with the intention of being used as a dataset decoder for further analysis or exploitation by the Analytics Engine. The MF files form the 2D array equivalent in JSON formatting. The second layer extends the meaning of the State 2D array elements from the Data Sample file for this particular domain. The first layer expresses the following:

Table 3 Attributes of the Mapping File in the IQU dataset

Label	Type	Description
Domain ID	unsigned int	The unique ID of the domain in the network
Pattern code	string	A code that includes various data about the element

The pattern codes are in String format, the letters used are unique and express a different attribute. The main patterns are expressed in Table 4:

Table 4 Meaning of the pattern codes

Pattern	Description
[0-9]	An integer located in the start of each code expresses the unique ID of each network element
-	The minus signed is used as a delimiter for word separation
cpu	CPU cores
ram	System memory
hdd	Storage
bw	Bandwidth
lat	Latency
s	When the letter “s” is present in a pattern it means that the following attribute “cpu/ram/hdd” is a service requirement or SLA
n	When the letter “n” is present in a pattern it means that the attribute is normalized and its value belongs in [0, 1] space

The patterns are designed to minimize the size of the files exchanged and can be easily decoded with the following example code snippet in Python:

```
if "attribute" in code:

The unique ID number can
code.split('-')[0]

Example Script:

state = [[0.1, 0.2, 0.3][0.4, 0.5, 0.6]]
code = {12: ["56-cpu-n", "56-ram-n", "56-hdd-n"], 18: ["57-cpu-n", "57-ram-n", "57-hdd-n"]}
state_i = 0
state_j = 0
for domain in code:
    for (domain_id, pattern) in domain:
        uid = pattern.split('-')[0]
        value = state[state_i][state_j]
        attribute = pattern.split('-')[1]
        normalisation = False
        if "n" in pattern.split('-')[2]:
            normalisation = True
        database.append(uid, domain_id, attribue, value, normalisation)
        state_j += 1
    state_i += 1
```

Figure 16 Sample pattern decoding script

The synthetic data is generated in real-time and the management of the network is taking place through the corresponding API or by using its respective Python module. All data can be converted to individual time-series for analysis by the AE, stored or broadcasted to other decision-making elements.

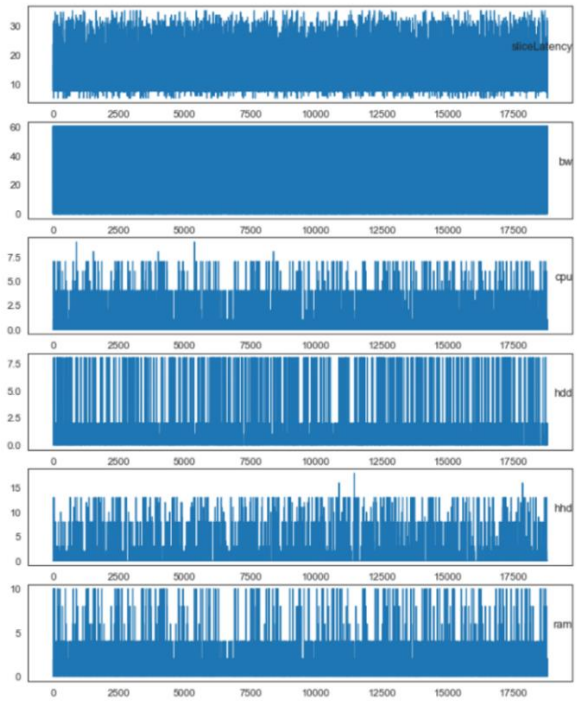


Figure 17 Computing resource fluctuation of a server during simulation

4.3.2 EUR DATASET

In this section, we illustrate the details of the EUR dataset.

Realistic 5G-data

The generated data is stored in a file with CSV structure. We dynamically allocate different resources (different RAM, CPU) to the AMF image and collect actual resources consumed by AMF while processing the received attach requests. This collected data is stored in a CSV file.

```
time,ram_limit,cpu_limit,ram_usage,cpu_usage,n,mean
1636553178,2048,2.000000,75722752.000000,0.003700,10,0.194402
1636553188,2048,2.000000,140255232.000000,0.003700,20,0.301621
1636553199,2048,2.000000,184238080.000000,0.052663,30,0.465659
1636553210,2048,2.000000,217681920.000000,0.052663,40,0.592062
1636553221,2048,2.000000,305606656.000000,0.120889,50,0.862765
1636553232,2048,2.000000,356622336.000000,0.120889,60,0.930602
1636553243,2048,2.000000,412884992.000000,0.249958,70,1.089852
1636553254,2048,2.000000,417267712.000000,0.249958,80,1.123452
1636553267,2048,2.000000,421892096.000000,0.399306,90,2.462652
1636553278,2048,2.000000,421892096.000000,0.240795,100,1.386847
1636553476,1024,2.000000,448122880.000000,0.005701,10,0.182663
1636553487,1024,2.000000,562274304.000000,0.000000,20,0.301439
1636553497,1024,2.000000,606273536.000000,0.016067,30,0.495922
1636553508,1024,2.000000,639688704.000000,0.016067,40,0.722723
1636553519,1024,2.000000,681287680.000000,0.000000,50,0.858051
1636553530,1024,2.000000,778563584.000000,0.105560,60,0.937769
1636553542,1024,2.000000,834224128.000000,0.331840,70,1.724131
1636553553,1024,2.000000,838496256.000000,0.246502,80,1.111750
1636553566,1024,2.000000,842719232.000000,0.347677,90,2.934816
1636553578,1024,2.000000,429604864.000000,0.347677,100,1.657437
1636553589,1024,2.000000,433360896.000000,0.340037,110,1.579026
1636553601,1024,2.000000,440102912.000000,0.486821,120,1.689424
1636553613,1024,2.000000,445677568.000000,0.486821,130,1.817818
1636553629,1024,2.000000,452784128.000000,0.573533,140,6.028824
1636553642,1024,2.000000,461975552.000000,0.539525,150,2.106529
1636553654,1024,2.000000,468828160.000000,0.456806,160,2.196627
1636553673,1024,2.000000,480653312.000000,0.531007,170,8.392036
1636553686,1024,2.000000,488517632.000000,0.519589,180,2.827986
1636553699,1024,2.000000,499134464.000000,0.431666,190,2.959742
1636553717,1024,2.000000,507129856.000000,0.674343,200,7.530571
```

Figure 18 Dataset structure of the EUR dataset

Column names

The data set consists of 28132 rows. Each row of the dataset contains a "**timestamp**", "**RAM Limit** ", "**CPU Limit**", "**RAM Usage**", "**CPU Usage**", the number of Attach Requests (noted as "**n**"), and the Attach Request duration (noted as "**mean**"). Table 5 presents the description of each column:

Table 5 Attributes of the EUR dataset

Columns names	Description

timestamp	The time of sending the attach requests
ram_Limit	The RAM allocated to the AMF image
cpu_Limit	The CPU allocated to the AMF image
ram_Usage	The RAM consumed by the AMF image
cpu_Usage	The CPU consumed by the AMF image
n	The number of Attach Requests
mean	the Attach Request duration (latency) in seconds. It corresponds to the time duration between sending an Attach Request and receiving the Registration Accept.

5 MonB5G MS/AE Architecture

5.1 Overview of the MonB5G Architecture

The main principle of the MonB5G architecture is to provide a framework for hierarchical, feedback-loop-based control for fault, configuration, accounting, performance, and security (FCAPS) management, and slice orchestration by using **different control loops with different scopes, goals, and timescales** at the Global OSS/BSS level, Technological/Orchestration Domain level, Slice level and Node (VNF/PNF/CNF) level. The MonB5G architecture consists of **static** (MonB5G Portal, Domain Manager and Orchestrator, Inter-domain Orchestrator, Infrastructure Domain Manager) **and dynamically deployed components** (slices i.e., a set of functions that implement a specific goal)**.** Moreover, an AI-driven In-Slice Management approach is adopted to separate the management functions of each slice and simplify slice management for slice tenants (e.g., through self-configuration, self-healing, performance optimization, exposure management interfaces etc.). Both the static and dynamic components of the MonB5G architecture follow the same management approach, as shown in Figure 19.

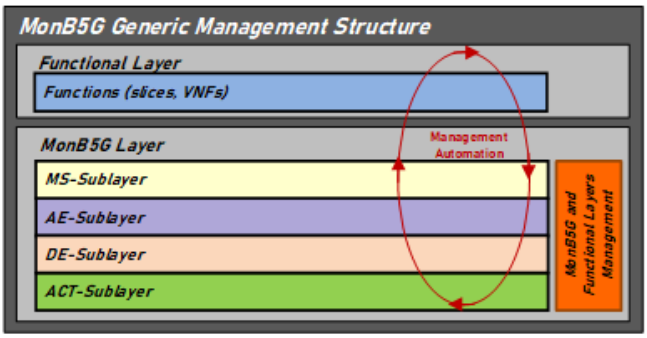


Figure 19 Generic view of MonB5G slice structure

The MonB5G generic management structure is composed of:

- **Monitoring Subsystem Sublayer** (MS Sublayer) – a set of functions responsible for collecting, aggregating and processing the monitored data as well as passing the results to other components of the architecture.
- **Analytic Engines Sublayer** (AE Sublayer) – several Analytic Engines focused on different goals, e.g. ,related to FCAPS.
- **Decision Engines Sublayer** (DE Sublayer) - multiple engines responsible for reconfiguration decisions based on data provided by system components.
- **Actuators Sublayer** (ACT Sublayer) – the components that transform DE decisions into atomic reconfiguration-related operations, with the goal of simplifying reconfiguration (by supporting abstractions and intent-based communication) and reducing management traffic between the DE and the reconfigured node(s).

The above mentioned sublayers enable the implementation of an AI-based MAPE management.
This section focuses exclusively on describing the generic architectures of MonB5G MS Sublayer and AE Sublayer. Note that, the internal structure, the characteristics of the processed data, as well as the scope of operations will strongly depend on the deployment site i.e., the parts responsible for slice orchestration (i.e., IDMO, DMO) will mainly use the data and domain-level slice KPIs provided by the domain orchestrators (e.g., MANO) , while at the

slice level (IDSM, network slice) the scope of monitoring and optimization will be more focused on the operation of a slice itself. It should be emphasized that further changes may occur due to the type of domain in which the MS/AE components are deployed (e.g., RAN, Cloud). Nevertheless, the implementation architecture of both MS and AE follows the generic principles, described in detail in D2.4 [38] so that they can be easily adapted or extended depending on the requirements derived from the specific deployment scenario (i.e., optimization goal, algorithms used, security requirements, etc.).

The generic structures of the **MS Sublayer** and the **AE Sublayer** are shown in Figure 20 and Figure 21, respectively. Both layers implement the publish-subscribe paradigm and expose the information to the entities higher in the hierarchy via a dedicated message bus.

The generic MS sublayer consists of:

- **Monitoring Information Collector/Aggregator** – data collection from Slice Functional Layer;
- **Monitoring Information Database** – a database containing raw and processed monitoring data;
- **Monitoring Information Processor** – an entity responsible for processing monitoring data;
- **Slice KPI calculator** – an entity that calculates (predicts) slice-specific KPIs;
- **Monitoring Sublayer Manager -** an entity that enables remote configuration of the MS sublayer.

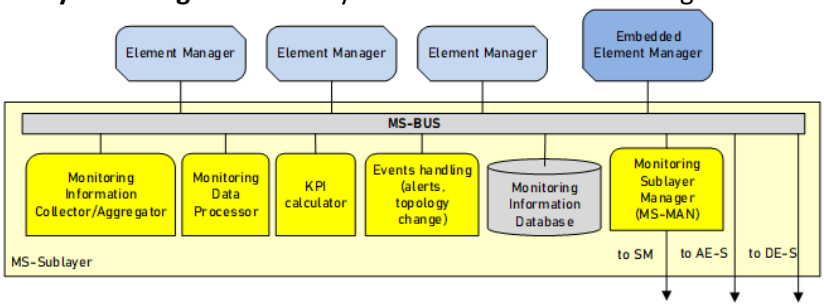


Figure 20 Monitoring System Sublayer internal components

In addition to collecting, aggregating, filtering, and interpolating monitoring data, MS is also responsible for calculating the slice KPIs and collecting information about faults and topology changes. The operations are performed with different temporal granularity and varying degrees of data aggregation, depending on the optimization goal.

The **AE Sublayer**, follows the same approach in that it includes a set of engines (AE) that perform singular analytic tasks, and the corresponding AE Sublayer Manager provides resources for managing each AE. Example analytic operations include security threat detection, real-time fault/performance degradation detection, etc. Analytics results are stored in a separate database.

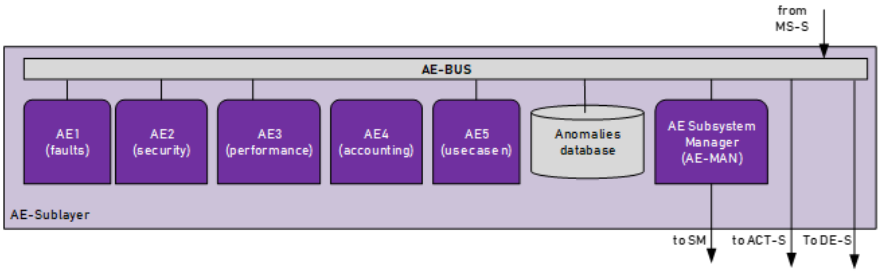


Figure 21 Internal components of Analytic Engine Sublayer

The following sections are dedicated to describing the implementation of the AEs or algorithms that can be easily adapted and reused when implementing the MonB5G architecture.

5.2 Architecture of MonB5G MS

The general design goals and principles of MonB5G MS were discussed in D3.1 [3]. In this section, we review the most important ones and then explain the architecture of the system in detail.

Based on the autonomic network management specifications and cloud-native design, MonB5G MS proposes a scalable architecture. The deployment of MonB5G MS follows the concept of Slice Management Layer as a Service proposed in the MonB5G architecture in D2.4 [38]. More specifically, MonB5G MS runs on top of a MonB5G Slice Management Layer (SML). SML's architectural characteristics enable the management of a single Slice Functional Layer (SFL) or multiple ones, in the form of SML as a Service or MonB5G Management Layer as a Service (MLaaS).

The SML underlay consists of NFV Objects, such as Virtual Network Functions (VNFs), orchestrated in close proximity to the managed SFL in each technological domain. These resources are then used to create a Platform as a Service (PaaS) in the form of a Container Infrastructure System (CIS), as described in ETSI NFV IFA 029. The PaaS instances are in turn managed by an Umbrella PaaS Controller, which according to MonB5G is placed at the centralized element location (e.g., OSS/BSS). Such an Umbrella PaaS Controller takes care of provisioning other PaaS instances (i.e., CIS) with specific services for a target MS/AE/DE component, even though AE/DE managers or functions may also request monitoring of some parameters.

The key concept of the MonB5G MS architecture is the *Sampling Loop*. Each loop is in fact a workload on CIS, periodically executing a Sampling Function. Each sampling function, as the name implies, takes a telemetry metric by contacting the Embedded Element Manager of the entities in the SFL. The monitoring system is responsible for creating, executing, and terminating the sampling loops.

MS runs within a CIS Instance and leverages a cloud-native design for messaging between its components, creates or modifies Sampling Functions, and makes collected telemetry data available to other services (i.e., AE, DE, or other MS instances). This enables autonomous TD management and minimizes the resource intensive exchange of raw telemetry data between TD of an end-to-end service.

MonB5G MS is essentially a cloud native application whose functionality is split into several microservices deployed on a CIS. The resources for the latter are orchestrated by the Domain Manager and Orchestrator (DMO), or the Central Element (e.g., IDMO). The components of MS in turn allow (authorized) external ***Requesters*** to launch, modify or delete Sampling Loops.

As mentioned earlier, sampling Loops are CIS workloads that execute a ***Sampling Function*** (i.e., one or more containers), whose Life Cycle Management (LCM) is delegated to the CIS itself. That is, MS interacts with the CIS control plane via its respective API to define the desired state of a particular Sampling Loop (specified by the Requester in a Sampling Loop configuration file), and CIS performs the tasks associated with Sampling Loop Deployment and LCM during its operational phase.

Figure 22 shows the MS architecture. To create Sampling Loops and provide the necessary infrastructure to exchange metrics in a streaming bus and databases, the MS implements several components:

1) **Manager:** admits the Sampling Loop configuration file, that defines the characteristics for the loop, including a reference to a Sampling Function. It also implements streaming bus Consumers, which are subscribers to a particular bus Topic and whose only task is to collect metrics and send them to the Time Series Data Base (TSDB).
    a. **Consumers (C_*):** are subscribers to a specific Topic. Their only task at Manager is to listen for metrics and then transfer them to the TSDB. Each Topic needs a Consumer at the Manager to transfer metrics to the TSDB.
2) **Sampling Function:** is the agent that implements the Sampling Loop Operations, i.e., triggers an API to collect metrics at a given interval and then passes them to the streaming bus. It is implemented as a Docker image. The Sampling Loop configuration file references SF e.g., via the Docker Hub registry.
3) **CIS API:** CIS control plane, i.e., Kubernetes API Server.
4) **Broker/Bus:** a streaming bus and broker e.g., Kafka.
5) **Time Series Data Base (TSDB):** a persistent storage populated by Consumers and used by other functions for bulk metrics retrieval.

The figure shows a Slice Functional Layer (SFL) and its components and an instance of a Slice Management Layer (SML) with a CIS control plane (**CIS API** in the figure). More specifically, MS is implemented in an isolated namespace within SML, and its components are interconnected via RESTful APIs, shown in this figure as different reference points (e.g., NBI and SBI).

- MS NBI: MS Northbound Interfaces include:
    - **m**: Loop creation HTTP API.
    - **q**: Kafka bootstrap. Used to subscribe/publish or create topic.
    - **db**: API endpoint for MS persistent storage.
        - Consumers use this NBI to populate the TSDB.
- MS SBI: MS Southbound Interfaces include:
    - **ka**: authorized service Account to CIS API.
    - **qsf**: Sampling Function (SF) access to Kafka bus.
    - **ms-e**: connector to Embedded Element Manager NBI.
- EEM NBI: this includes:
    - **eem-nbi**: exposed metrics from Monitored Element.

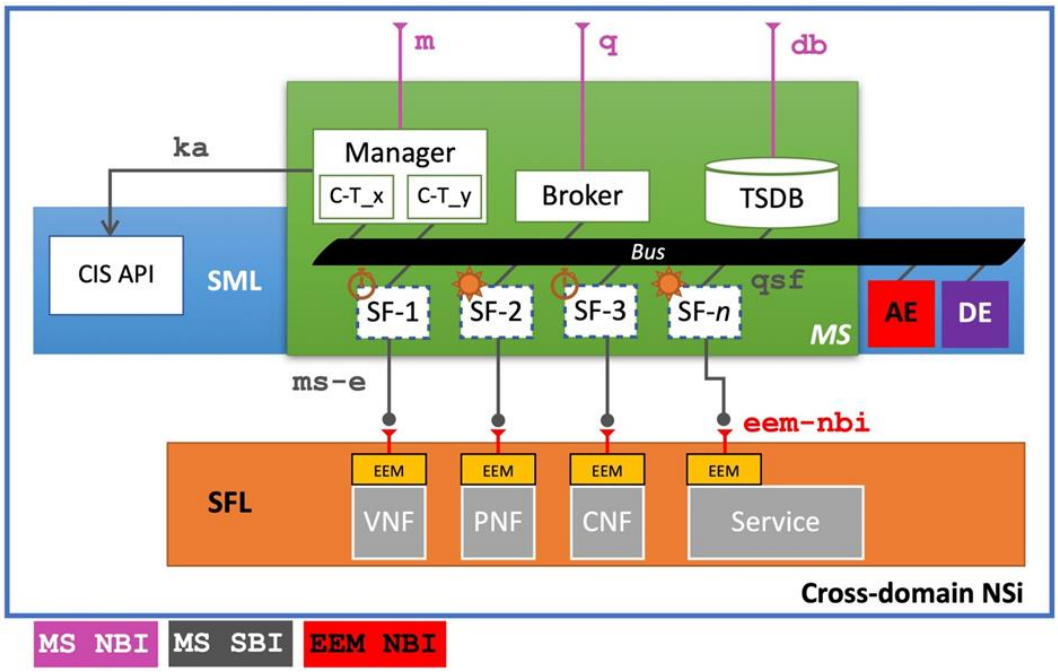


Figure 22 MonB5G MS internal components and reference points[33]

In summary, the distributed architecture of the MonB5G monitoring system provides the following advantages. First, the distributed MS agents are designed to manage the tightest metric sampling loops in their respective technological domains, largely reducing the need for data transmission and thus minimizing the communication overhead introduced by the monitoring system itself. Furthermore, an additional MAPE-based embedded element manager is deployed at the VNF level to support fine granularity (1s) of telemetry acquisition. It also enables the development of aggregators for specific (e.g., slice-level) AE and DE. More importantly the configurations of MS entities distributed across different technical domains are automatically defined and triggered by the AE/DE components with AI-assisted policy-driven mechanisms, which is a crucial step towards highly automated slice-level monitoring system.

5.3 Architecture of MonB5G AE

As show in Figure 23, MonB5G AE has been designed to contain two main functions, namely, KPI prediction and fault detection, and exploits MonB5G distributed architecture to push the analysis close to the data collected by MS in each domain (i.e., RAN, Edge and Cloud), minimizing the need to transfer raw slice performance and configuration data across the different network domains and slices. This leads to a dramatic reduction in transmission overhead and yields more scalability in managing a massive number of slices.

[33] CIS API interface to Umbrella PaaS Controller is implemented leveraging Kubernetes KubeFed library: https://github.com/kubernetes-sigs/kubefed

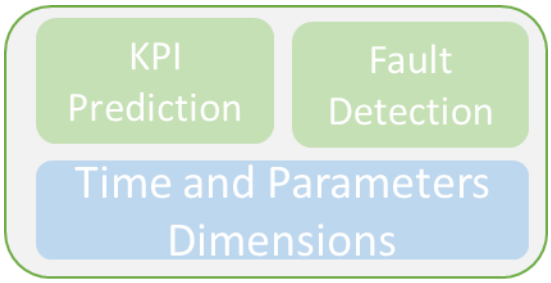


Figure 23 Two major functions of MonB5G AE

The function of KPI prediction is designed to perform:

Time-series prediction of slice-level metrics such as the traffic and resource usage in order to help the Decision Engine (DE) taking preventive actions against e.g., Service-Level Agreement (SLA) violations. To that end, time-series prediction leverages customized and fine-tuned serial architectures such as Long Short-Term Memories (LSTM) or advanced variants.

Parameters' space prediction, which aims at establishing accurate models to link certain measured input metrics (e.g., traffic per slice, channel quality, CPU load) with a target output metric (e.g., energy consumption per slice) while ensuring a certain SLA requirement. This type of analysis will guide the DE to fine-tune its action space, where it can know the order of magnitude/interval of the action to achieve e.g., a low SLA violation rate. To that end, AE is intended to adopt a new class of statistical/constrained neural networks/models, among other techniques.

The function of fault detection is to detect abnormal events during a slice lifecycle, by mainly extracting and recognizing changes in data distributions and trends. This relies on advanced techniques of clustering and classification based on novel architectures of neural networks. Since slices are typically software and virtualization-based, the notion of fault includes but not limited to the infrastructure on top of which the slice is running. Specifically, a slice fault could be a logical abnormality where, e.g., the classification of slice traffic (inspired by deep packet inspection) reveals that it does not fit into the slice predefined template, and therefore the isolation is breached.

5.4 Cooperation between MS, AE and DE

Traditional centralized network management is a bottleneck and cannot scale with massive numbers of network slices expected in beyond 5G and pre-6G cellular systems. Centralized approaches can benefit from a holistic view of the entire network, but scale poorly in realistic scenarios and incur a significant monitoring overhead. Therefore, distributed ML techniques are imperative to:

- minimize the exchange of information in the network
- ensure the scalability of the entire slice management system
- reduce the latency time.

The goal of this section is to provide an overview of the communication links and interactions between the (local/central) MS and the AE and the DE blocks that have been proposed as part of MonB5G to illustrate the interaction between these components.

Figure 24 shows how the MS, AE and DE blocks are intended to communicate with each other. This figure also includes the actuators that translate DEs decisions into API calls to different slice components (e.g., VNFs, links, PNFs) in each of the technological domains (RAN, Edge, Cloud) that a slice should traverse.

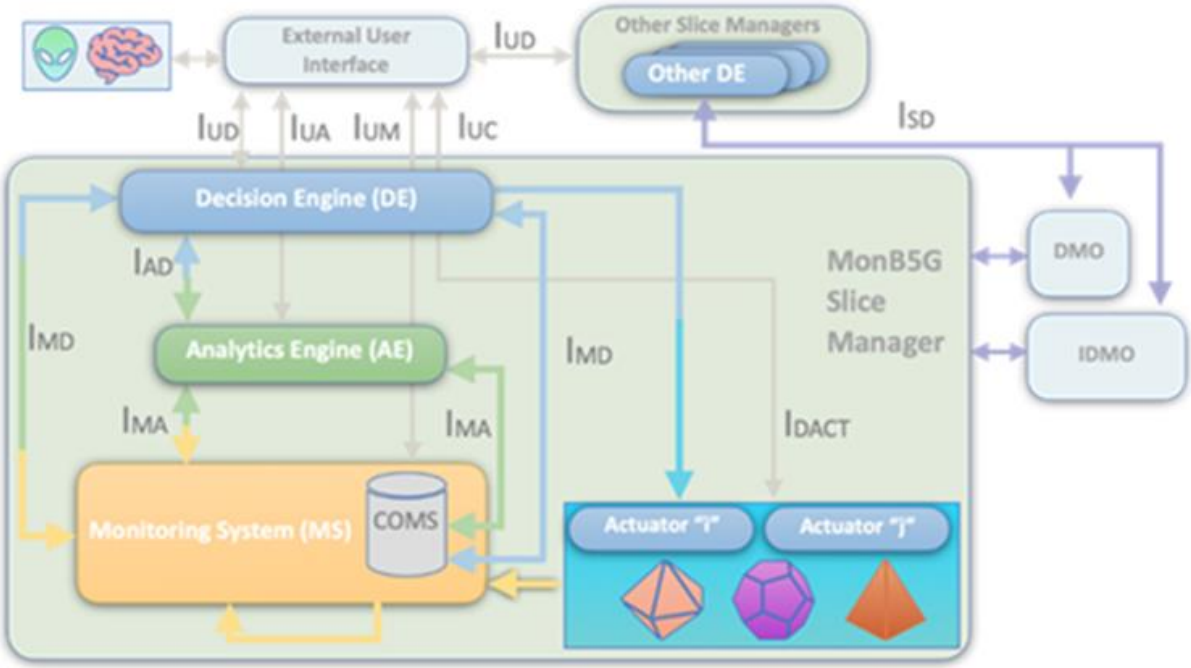


Figure 24 Interfaces related to MS and AE

The MS in Figure 24 is the entity responsible for collecting a number of different metrics from the systems that the DE controls. This information can be passed directly to DE and AE, but it is also stored in a common online memory store (COMS), represented by the grey cylinder in Figure 24. This COMS was added to avoid hard synchronization conditions between MS, DE and AE when information needs to be exchanged. In this way, DE and AE can be more flexible in terms of the duration of their processing without compromising the granularity with which MS can retrieve monitoring data from the controlled systems. Thus, it is the MS (depending on its capabilities and the amount of information as well as the granularity set by the External User Interface (EUI)) that somehow defines how fast the data is sampled. Note that COMS coincides with the 'Knowledge' block of the presented ETSI ZSM functional scheme.

The AE then reads the monitoring information from the COMS to preprocess it (e.g., perform predictions) before making it available to the DE. The AE can also read the information directly from the MS, but this is likely to be used in more selective cases, where some synchronization is required. The prediction interval can also be set via the EUI. Once the AE outputs the preprocessed data, it stores it in the COMS.

Similarly, DE is expected to read its input from COMS. However, it is also possible to obtain this information directly from AE and MS. Once the DE has generated its decisions, it also stores them in the COMS and passes them to the actuator interfaces of the systems it controls to translate them into API calls for slice components lifecycle management (LCM). DE parameters can also be fine-tuned at runtime by the EUI and may take effect in the next DE configuration update interval.

The following two tables summarize descriptions of the various interfaces that connect AE and MS to other MonB5G entities.

Table 6 MS Interfaces and the Associated Roles

Interface	Type	Role
I_{MD}	Tensors/Database query	DE reads raw MS measurements (either online or from COMS)/Store AI metrics and DE decisions in COMS
I_{MA}	Tensors/Database query	AE reads raw MS measurements/Store AI metrics, predictions in COMS
I_{UM}	Database Query	EUI reads/changes MS configuration (e.g., granularity)

Table 7 AE Interfaces and the Associated Roles

Interface	Type	Role
I_{AD}	Tensors/Database query	DE Reads the predicted KPI from AE (either online or from COMS)
I_{MA}	Tensors/Database query	AE reads raw MS measurements/Store AI metrics, predictions in COMS

6 Distributed MonB5G Monitoring System

This chapter discusses the details of the operation of the monitoring system. The architecture of the system was explained in Section 5.2. In this chapter, we explain how the architecture is implemented and used in practice. At the beginning, the steps of the workflow of the system are explained. The use of MS in both single and different technical domains is the next section of this chapter. Finally, after discussing the telemetry monitored by MonB5G MS, the details of the implementation of the system are explained.

6.1 Workflow of MonB5G MS

As explained earlier, the main task of MS is to receive the monitoring requests from the requesters, launch the corresponding sampling loops, and store the monitoring data in the database or streaming bus. This workflow is illustrated in Figure 25, which includes a Message Sequence Diagram (MSD) detailing the procedure for setting up a Sampling Loop.

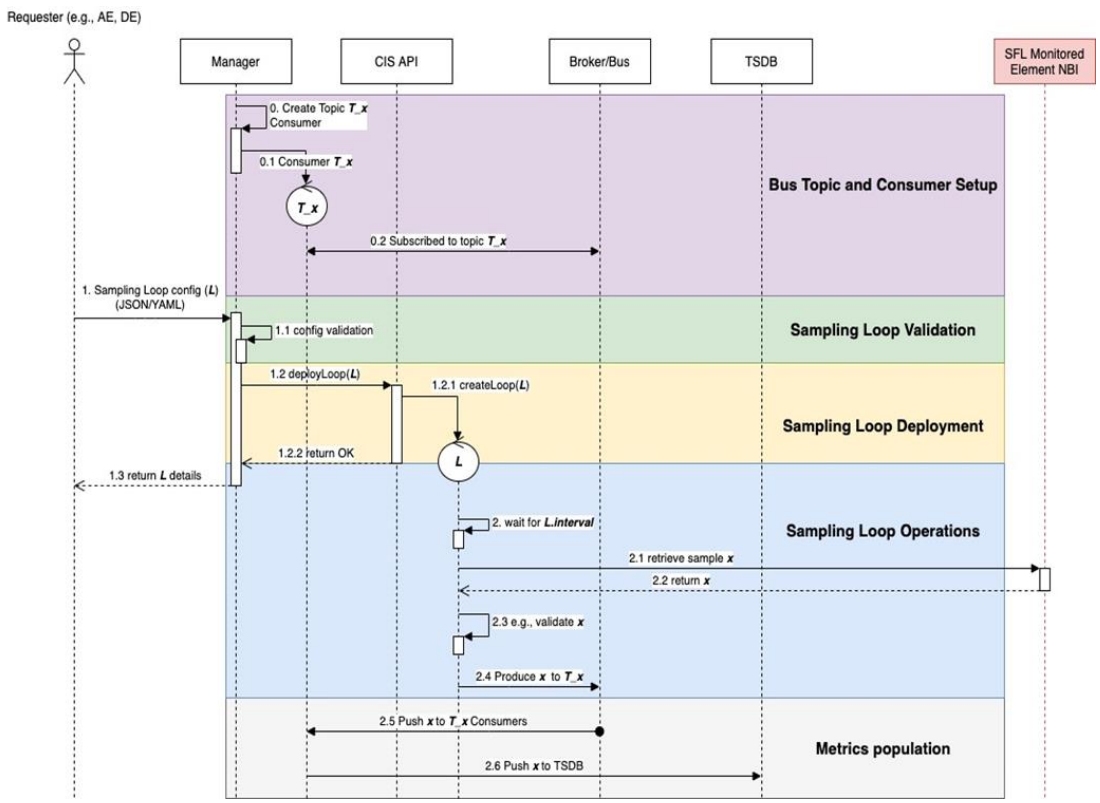


Figure 25 Generic Message Sequence Diagram for the Request of a Sampling Loop to a MS instance

As can be seen in Figure 25, this workflow is composed of the following steps:

i. Within the Manager component of MS, a set of (Kafka) streaming Consumers is configured.

1.1 These are subscribed to preconfigured topics, which are used to populate metrics.

1.2 Messages produced in a particular Topic (e.g., T_x) are sent to the subscribed Consumers.

ii. A Requester sends a Sampling Loop configuration file (L) to Manager's Northbound Interface (NBI). This configuration contains the Sampling Function, the sampling Interval, the NBI address of Monitored Element, and the Topic.

2.1 Manager performs a basic syntax check of L.

2.2 Manager populates a CIS Workload Template with data from the Sampling Loop configuration file.

2.2.1 Via CIS API, a custom CSI Workload is deployed that represents a Sampling Loop is deployed.

2.2.2 A confirmation is returned to Manager.

2.3 The details of the deployed Sampling Loop are returned to Requester.

iii. Each loop boots up and waits for an interval specified in the Sampling Loop configuration. This interval determines the frequency of sampling.

3.1 After the interval, the Sampling Loop (i.e., its Sampling Function) triggers the Monitored Element NBI (i.e., its Embedded Element Manager (EEM)).

3.2 A metric x sample is retrieved.

3.3 Later, validation or other pre-processing is performed (this applies to any Sampling Function).

3.4 Finally, the sample x is sent by the Producers in the Sampling Function to the streaming bus with topic name T_x.

4.4 The consumers at the Manager of the topic T_x forwards the sample x to the Time Series Data Base instance in the Technological Domain (TD).

Note that the workflow described in Figure 25 can be triggered by any authorized Requester e.g., Tenant, DMO, AE, or DE functions, among others.

6.2 MonB5G MS at Different Technical Domains

Since supporting multiple domains is one of the main goals of the MonB5G architecture, the monitoring system was also designed and implemented to support multiple domains. This subsection details the development of the distributed monitoring system for several technological domains. For the RAN, Edge and Cloud domains, we illustrate the locally deployed cloud-native MS entities with comprehensive discussions of their technical KPIs, including data collection granularity, computation and storage efficiency and cost and communication overhead. Although the MS entities are distributed across different technical domains, they are designed for zero-touch management and orchestration with the MonB5G platform. The domain-specific MS entities follow the following main concerns:

- Stay in line with the MonB5G framework and target KPIs

- Meet functionalities, including data collection, aggregation, data pre-processing and storage
- Implement different granularities for non-real-time, near-real-time and real-time intelligent control
- Promote flexibility of MS entities whose activities are triggered and configured by AEs and DEs
- Promote scalability and efficiency, e.g., reduce communication overhead

6.2.1 *MS IN RAN*

The MonB5G monitoring system inspects the RAN domain as shown in Figure 26. It constitutes MS-sublayer of the SML for the RAN domain, and consists of the following functional components:

- **The Monitoring manager (MM)** is responsible for lifecycle management: triggering, managing and deleting a monitoring task. It also supports remote configuration of MS operations.
- **The Monitoring data collector (MDC)** connects to the RAN management and orchestration system to collect detailed real-time metrics from eNB/gNB.
- **The Monitoring data processor** performs data pre-processing functions defined by the analytics engine requirements, such as aggregation, filtering and normalization. This helps to further reduce the communication and computation overhead.
- **The Monitoring database (optional)** is used to store raw telemetry data. The stored historical data facilitates the improvement of AI models used in AEs and DEs.
- **The MS-bus** is used for real-time data transfer between the above components. The publish/subscribe tools, such as Kafka[34], can be used for high-throughput, low-latency unified communication.

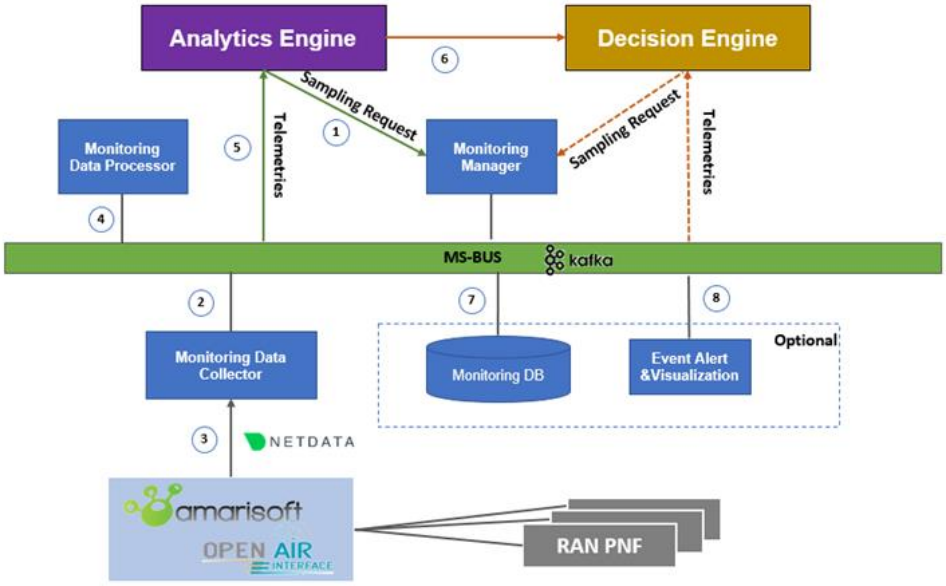


Figure 26 MonB5G MS in the RAN Domain

[34] https://kafka.apache.org/

6.2.2 *MS IN EDGE*

The monitoring systems in the edge and cloud domains shown in Figure 27, are almost similar to the architecture of the monitoring system for RAN, except that the data is collected using the VNFs and computing infrastructure instead of the RAN PNFs. They both run on PaaS and deploy MS components as sampling functions. The MonB5G platform aims for zero-touch management and orchestration. The MS modules are automatically created, maintained, and used by AEs and DEs based on the actual status of the network slices. AEs and DEs can query a variety of telemetry data. Therefore, we propose the following MS module deployed in MEC/cloud to collect telemetry data from VNFs.

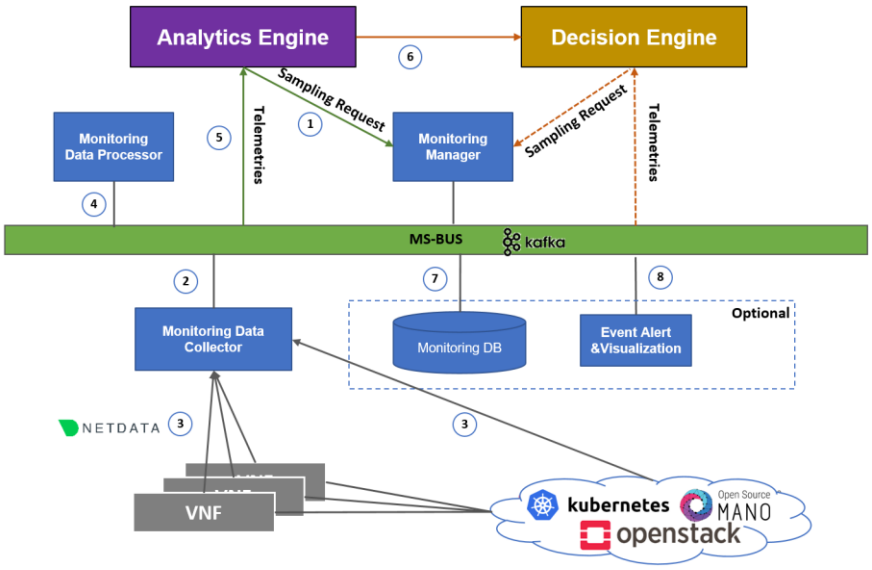


Figure 27 MonB5G MS in the Edge/Cloud Domain

The main functions of the MS components, such as the monitoring manager and the monitoring data processor, are implemented similarly to those of the RAN domain. The main differences are in the monitoring data collector component. In edge/cloud domain, each instantiated VNF is associated with a Netdata instance. When AE requests telemetry data from specific VNFs for analysis, the monitoring data collector activates the corresponding Netdata instances associated with the VNFs to provide the telemetry data based on the configurations defined by AE. MS is structured as a sublayer located in the slice management layer of the MonB5G platform, and all FCAPS functions can be dynamically deployed or updated during the slice lifetime using the orchestration capabilities of Inter-Slice Management (ISM), which also support MS with the resource scaling mechanism.

In summary, the distributed MonB5G MS captures the operational status at multiple levels of the management hierarchy (node, slice, domain, and inter-domain). Once triggered and configured by AEs, the programmable MS entities connect the appropriate infrastructure and network functions (VNFs and PNFs) to collect the requested telemetry data.

6.3 Telemetry Monitored by MonB5G MS

MonB5G Monitoring system is a generic platform that can theoretically collect all measurable telemetry data. As explained in the previous section, the telemetry data is measured by the environment through the corresponding EEM (for example, Netdata measures the CPU or RAM utilisation). The corresponding sampling function requests the data from the EEM and publishes it on the bus in MS. As long as there is an EEM for telemetry data that exposes the data via a (property) protocol, it is possible to develop the corresponding sampling function and consequently use it in the monitoring system. In other words, the list of telemetry data is determined by the EEM (located external to the SM) and not by the monitoring system itself. To further explain the telemetry data to be monitored, Table 8 shows the list of metrics and their mapping to the platform.

Table 8 Slice-Level KPIs Description and Measurement

Metric	Description	Category	Type	Mapping to the Platform
Latency (control plane (CP) and user plane (UP))	Average E2E downlink packet delay between Core cloud and UE	Integrity	Mean	End-to-end latency must be explicitly measured independently of CP/UP. That is, the MonB5G administration elements should send control packets (timestamped) through such planes and calculate the delay. For CP, MS is expected to serve as a source/sink for such timestamped packets. UP delay estimations on the other hand, would require the UE to send these aforementioned control packets to derive an accurate latency estimate at RAN while UP at the Edge/Cloud TD latency can be derived by MonB5G administrative elements in a similar manner as at the CP.
Throughput	Total packets Bytes divided by the granularity period	Integrity	CumSum	Throughput can be derived from per-VNF telemetry. That is, by measuring the number of packets traversing the VNF interfaces (e.g., using EEM such as NetData), an accurate estimate of available throughput for the entire slice can be derived.
CPU usage	CPU consumption/Available CPU capacity	Usage	Ratio	The consumption of Per-VNF CPU and RAM can be queried using EEM (e.g., via the API of the NetData instance).
RAM usage	RAM consumption/Available RAM capacity	Usage	Ratio	
PRBs usage	PRB consumption/Total PRBs per slice	Usage	Ratio	Wireless SDN Controllers from which the number of RB dedicated in the Uplink/Downlink direction can be queried/configured.
Number of admitted slices	Number of slices successfully admitted with resource allocation fulfilling SLA	Accessibility	CumSum	By using NFVO Northbound interfaces (NBI) (e.g., Os-Ma), OSS/BSS (or other MonB5G administrative components) can query the number of running (or failed) network slice instances.
Connected users/slice	Number of RRC connected users per slice	Accessibility	CumSum	Users are directly mapped to UEs subscribed to an e/gNodeB. This metric can be extracted at the RAN

				management level (using its NBI) or by extracting this information from 4GvEPCor 5GC.
Slice end-to-end availability time	Average time with slice VNFs is all available	Retainability	Mean	Slice 'readiness' is announced by MonB5G administrative components to DMO via the domain shared messaging bus.
VNF/PNF availability time	Average time where a VNF/PNF is not in fault	Retainability	Mean	
Reliability (Packet loss rate)	Number of erroneous packets/total packets per slice	Integrity	Ratio	Packets may be lost in the RAN or NFV segments of the Slice. Therefore, RAN controller's NBI and VNF-instances' telemetry API (i.e., EEM) are used to compute this metric.
Data volume exchanged between the local and end-to-end entities	Total Bytes exchanged between a local and end-to-end entities	Integrity	CumSum	MonB5G administrative components should keep accounting of Requests/Responses as well as Publish/Subscribe operations between technological domains. This bookkeeping can be queried directly via their respective NBI and serves as input for the calculation of this metric.
Consumed energy	Average Power consumption x granularity period	Integrity	Mean	Originally this was expected to be the result of calculations that use CPU/RAM consumption as the base metric.
Number of operations	Total slice life-cycle management operation (scaling, placement, termination, etc) in a granularity period	Integrity	CumSum	At Network Slice Instance level, this metric can be queried using NFVO NBI. Such operations can be: VNF scaling, termination, etc.

6.4 MonB5G MS Implementation and Visualization

This section discusses some important details of implementation of MonB5G MS and also present some real implementation results.

6.4.1 IMPLEMENTATION DETAILS

In the previous sections, it was discussed that the distributed architecture of MonB5G MS is implemented on the CIS across multiple technological domains. In this section, we describe more details of the implementation.

The monitoring system is deployed as a Kubernetes cluster, i.e., the components of the system, including the "manager" and "sampling functions", are actually implemented as pods on the Kubernetes cluster. There are four types of pods in the system:

- **Manager pod**: there is only one pod of this type in the current implementation (it may be replicated in future versions). This pod consists of two containers, i.e., the "manager container" and the "Kafka-

consumer" container. The manager container implements the "m" interface and is responsible for LCM of the sampling functions. The Kafka consumer container is responsible for copying the telemetry data into the TSDB.

- **Kafka pod**: there are several pods of this type; they implement the bus and the broker of the architecture; this pod consists of multiple "zookeeper" and "broker" containers.
- **TSDB pod**: this pod implements the TSDB, which are in fact the containers of the influxdb system.
- **SF pod**: there are multiple pods of this type. Each pod implements a sampling function consisting of two containers: sampler container and Kafka producer container. The sampler container implements the eem-nbi interface and communicates with the EEM to get the telemetry data. The Kafka producer receives the telemetry data from the sampler container via the virtual file system of the pod and publishes it on the bus.

The data flow through these system components is shown in Figure 28. In step (1), data is retrieved from the EEM by the sampler container. This container extracts the data from the eem-nbi interface and writes it to a file in the virtual file system of the SF pod, which is step (2). The Kafka producer container retrieves the data from the virtual file system and publishes it to the bus in step (3) under the topic specified in the configuration of the sampling loop. The Kafka consumer container in the manager pod has already subscribed to this topic and therefore receives the data in step (4). This container stores this data in the TSDB in step (5). Finally, the sampled data is delivered to the external application such as AE or DE. If these applications have already implemented the messaging protocol, they can connect directly to the bus to retrieve the data. Otherwise, this data is available through the TSDB.

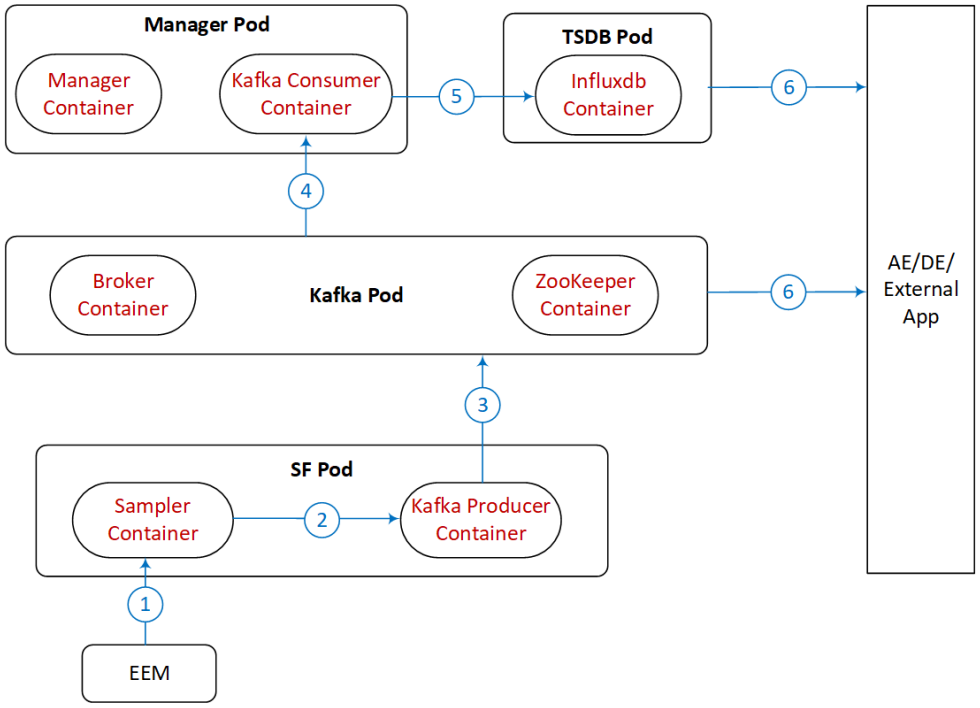


Figure 28 Data flow in the monitoring system

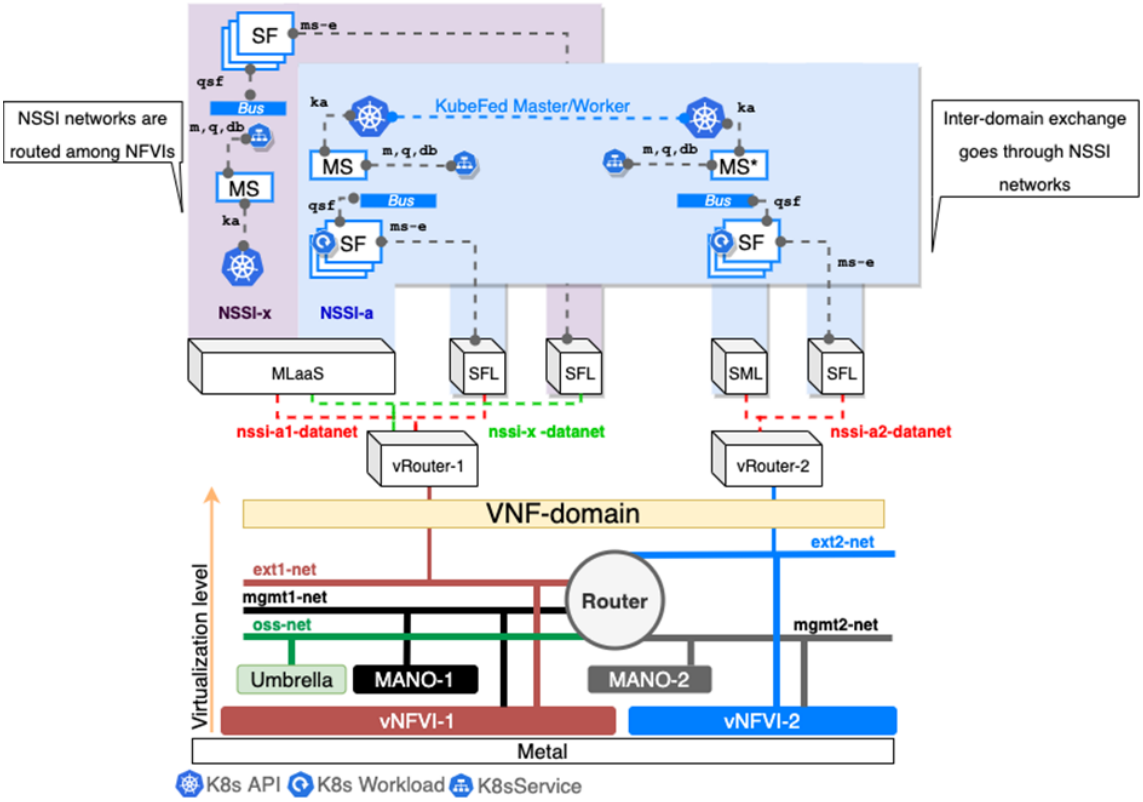


Figure 29 Implementation view of MS on two Technological Domains

Figure 29 depicts the implementation of the MS in two domains. This figure shows a view of SML and MS, serving two NSSIs, one of which (NSSI-a) has SFL components on two Technological Domains (TD i.e., vNFVI-1 and vNFVI-2, emulating Cloud and Edge domains). As shown, each SFL component has access to external *Provider Networks* (i.e., nssi-*-datanet) that enable communication between SFL/SML component seven though they are located on different TD. Upon request (via ***m*** reference point), MS creates specific SF in close proximity to the SFL components, reducing latency in retrieving telemetry data and limiting MS traffic to its TD.

To support multi-domain monitoring, the ***KubeFed*** control plane between the two K8s APIs (i.e., CIS API) enables the deployment of MS components (*MS** in Figure 29) on a target CIS (*SML* in Figure 29), resulting in LCM of such workloads. This process requires the implementation of a new field in the Sampling Loop configuration file that specifies the Sampling Loop and the *Destination CIS*. Manager then uses the reference point ***ka*** and ***KubeFed*** to request the creation of Sampling Loops at specific CIS.

The CIS i.e., K8s, used to implement MS has limitations in orchestrating Sampling Loops with periods of less than one minute; therefore, sampling loops are divided into two types: *1)* periodic executions of Sampling Loop Orchestration + Operations managed by CIS, or *2)* a single execution with embedded configuration to ensure sufficient sampling resolution. Type-*1* Sampling Loops are implemented as Kubernetes CronJobs, while type-*2* are standard Deployments. This is illustrated in Figure 30.

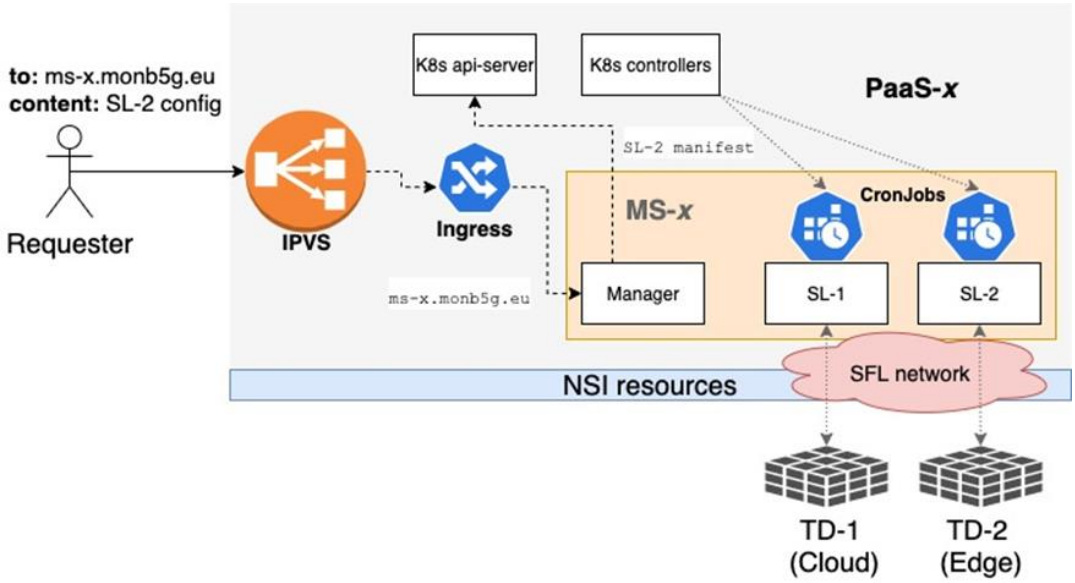


Figure 30 Mapping Sampling Loop Orchestration to Implementation Tools

Sampling Loop configuration files should follow the MonB5G information model for MS. That is, it is a configuration file for a known API designed for the configurations supported by MS (e.g., CIS workload type, TD, sample retrieval method, etc.). In addition, CIS workloads require that Sampling Functions need to be designed as Docker images that allow configuration parameters via predefined environment variables.

Figure 30 shows an example of a Sampling Loop configuration file. In summary, it specifies the following:

- An array of sampling loops (`.config.loops`)
- Each loop should in turn define:
  - `.config.loops.[*].name`: a unique name.
  - `.config.loops.[*].container`: contains a reference to:
    - `dockerImage`: SF itself.
    - `args`: specific arguments that can be passed to SF.
    - `env`: adds environment variables to SF, to apply orchestration-time configurations. These include:
      - `TARGET`: the eem-nbi endpoint.
      - `INTERVAL`: sampling interval in seconds.
      - `TOPIC`: the Kafka topic in which the results of this SF will be published
    - `resources`: the maximum *CPU* and *RAM* resources allocated to a given SF.

```
{
    "config": {
        "loops": [
            {
                "name": "cpu01",
                "container": {
                    "dockerImage": "bahadorbakhshi/netdata_sf:final",
                    "args": [],
                    "env": [
                        {
                            "name": "TARGET",
                            "value": "http://10.64.116.10:19999/api/v1/data?chart=system.cpu
&after=-1&format=json"
                        },
                        {
                            "name": "INTERVAL",
                            "value": "2"
                        },
                        {
                            "name": "TOPIC",
                            "value": "ms.cpu"
                        }
                    ],
                    "resources": {
                        "limits": {
                            "memory": "64Mi",
                            "cpu": "250m"
                        }
                    }
                }
            }
        ]
    }
}
```

Figure 31 A Sampling Loop configuration

As mentioned earlier, the sampling function, which will be dockerized, implements the eem-nbi interface and communicates periodically with the EEM. An example of the implementation of a sampling function for Netdata EEM is shown in Figure 32.

```
from datetime import datetime
import requests
import pprint
import json
import os
import time
import logging
from logging.handlers import RotatingFileHandler
import urllib

TARGET = str(os.environ.get("TARGET"))
INTERVAL = int(os.environ.get("INTERVAL"))
LOG_DIR = str(os.environ.get("LOG_DIR"))

targets = list(TARGET.split(','))

while True:
    for t in targets:
        payload = {}
        headers = {}
        response = requests.request("GET", t, headers=headers, data=payload)

        res = dict(response.json())
        target_ip = str(urllib.parse.urlsplit(t).netloc).split(':')[0]
        res.setdefault("target", target_ip)
        res.setdefault("sample_time", datetime.now().timestamp())

        pprint.pprint(res)

        with open(LOG_DIR + "/" + str(datetime.now()), 'w') as output:
            output.write(json.dumps(res))
        output.close()

    time.sleep(INTERVAL)
```

Figure 32 Netdata Sampling Function

Kubernetes passes the environment variables TARGET, INTERVAL, and LOG_DIR to the sampling function. Then, the sampling function connects to the TARGET, retrieves the telemetry data, and stores the responded telemetry data in a separated file in the LOG_DIR every INTERVAL second. These files are processed by the Kafka producer container of the pod and their contents are published to the bus.

```
from pprint import pprint
from kafka import KafkaConsumer
import pprint
import json
import os

BOOTSTRAP = $IP_FROM_KAFKA_BOOTSTRAP
TOPIC = 'ms-netdata.*'
TOPIC_REFRESH_PERIOD = 5000  # ms between KafkaConsumer topicList refresh

def start():
    consumer = KafkaConsumer(group_id='an-id-for-each-consumer', bootstrap_servers=[BOOTSTRAP],
        value_deserializer=lambda m: json.loads(m), auto_offset_reset='earliest',
        enable_auto_commit=False, metadata_max_age_ms=TOPIC_REFRESH_PERIOD)

    consumer.subscribe(pattern=TOPIC)

    consumer.poll()
    consumer.seek_to_end()

    for m in consumer:
        raw_JSON = m.value

        t = str(m.topic)
        if t == 'ms-netdata.cpu':
            print("[CPU]")
        elif t == 'ms-netdata.ipv4errors':
            print("[IPV4 ERRORS]")
        else:
            "NO TOPIC"
            continue
        pprint.pprint(raw_JSON, indent=4)


if __name__ == '__main__':
    start()
```

Figure 33 MS Kafka Consumer example (Python)

Requesters (e.g., users, MS, AE, or DE) can upload, create, or destroy various Sampling Loop configurations on demand. Moreover, requesters can implement a MS Consumer such as the one in the example Figure 33 to receive telemetry data collected by the Sampling Functions.

6.4.2 IMPLEMENTATION RESULTS

In this section, some sample results of the real deployment of the monitoring system in the CTTC testbed are presented. The following figures show some results of the sampling loop deployed by the configuration shown in Figure 31. These results correspond to the points (2), (3), (5) and (6) in Figure 28.

Figure 34 corresponds to point (2) in Figure 28showing two Netdata telemetry data sampled and reported by the sampling function. The returned data from the EEM is a JSON showing the different portions of the CPU consumption, e.g., “user”, “system” and “IRQ”.

```
{'data': [[1650364219,
           0,
           0,
           0.1274697,
           0.3186743,
           0,
           4.206501,
           2.676864,
           0.1912046,
           0.0637349]],
 'labels': ['time',
            'guest_nice',
            'guest',
            'steal',
            'softirq',
            'irq',
            'user',
            'system',
            'nice',
            'iowait'],
 'sample_time': 1650364220.061012,
 'target': '10.64.116.10'}
{'data': [[1650364224,
           0,
           0,
           0.2567394,
           0.449294,
           0,
           5.712452,
           2.439024,
           0.1283697,
           0]],
 'labels': ['time',
            'guest_nice',
            'guest',
            'steal',
            'softirq',
            'irq',
            'user',
            'system',
            'nice',
            'iowait'],
 'sample_time': 1650364225.078173,
 'target': '10.64.116.10'}
```


Figure 34 Two Netdata samples in point (2) of Figure 28

This sampled telemetry data is passed to the Kafka producer container which publishes them on the bus for the specified topic in the loop configuration. Figure 35 shows the output of this container, which represents the data not only in JSON format, but also in byte streaming, which is serialized to be published in the bus.

```
-----------------------
Reading data from /var/log/2022-04-19 10:30:20.061931
Sending data to Kafka ...
data: {"labels": ["time", "guest_nice", "guest", "steal", "softirq", "irq", "user", "system", "nice", "iowait"], "data": [[1650364219
, 0, 0, 0.1274697, 0.3186743, 0, 4.206501, 2.676864, 0.1912046, 0.0637349]], "target": "10.64.116.10", "sample_time": 1650364220.0610
12}
TOPIC:ms.cpu
data :{"labels": ["time", "guest_nice", "guest", "steal", "softirq", "irq", "user", "system", "nice", "iowait"], "data": [[1650364219
, 0, 0, 0.1274697, 0.3186743, 0, 4.206501, 2.676864, 0.1912046, 0.0637349]], "target": "10.64.116.10", "sample_time": 1650364220.0610
12}
dump :b'"{\\"labels\\": [\\"time\\", \\"guest_nice\\", \\"guest\\", \\"steal\\", \\"softirq\\", \\"irq\\", \\"user\\", \\"system\\",
\\"nice\\", \\"iowait\\"], \\"data\\": [[1650364219, 0, 0, 0.1274697, 0.3186743, 0, 4.206501, 2.676864, 0.1912046, 0.0637349]], \\"ta
rget\\": \\"10.64.116.10\\", \\"sample_time\\": 1650364220.061012}"'
medata.topic:     ms.cpu
medata.partition: 0
medata.offset:    236
-----------------------
Reading data from /var/log/2022-04-19 10:30:25.079240
Sending data to Kafka ...
data: {"labels": ["time", "guest_nice", "guest", "steal", "softirq", "irq", "user", "system", "nice", "iowait"], "data": [[1650364224
, 0, 0, 0.2567394, 0.449294, 0, 5.712452, 2.439024, 0.1283697, 0]], "target": "10.64.116.10", "sample_time": 1650364225.078173}
TOPIC:ms.cpu
data :{"labels": ["time", "guest_nice", "guest", "steal", "softirq", "irq", "user", "system", "nice", "iowait"], "data": [[1650364224
, 0, 0, 0.2567394, 0.449294, 0, 5.712452, 2.439024, 0.1283697, 0]], "target": "10.64.116.10", "sample_time": 1650364225.078173}
dump :b'"{\\"labels\\": [\\"time\\", \\"guest_nice\\", \\"guest\\", \\"steal\\", \\"softirq\\", \\"irq\\", \\"user\\", \\"system\\",
\\"nice\\", \\"iowait\\"], \\"data\\": [[1650364224, 0, 0, 0.2567394, 0.449294, 0, 5.712452, 2.439024, 0.1283697, 0]], \\"target\\":
\\"10.64.116.10\\", \\"sample_time\\": 1650364225.078173}"'
medata.topic:     ms.cpu
medata.partition: 0
medata.offset:    237
```

Figure 35 Two Netdata samples in Figure 34 which are in point (3) of Figure 28

As mentioned earlier, the data published on the bus, is not only directly accessible to AE/DE, connected to the broker, but it is also stored in the time series database by the Kafka consumer that received the data from the bus and wrote it to the TSDB. This happens in point (5) of Figure 28. The output history of the container is shown in Figure 36.

```
236 - [Topic: ms.cpu]
data_record: 2022-04-19\ 10:30:35.045800,topic=ms.cpu fields="{\"labels\": [\"time\", \"gues
\", \"irq\", \"user\", \"system\", \"nice\", \"iowait\"], \"data\": [[1650364219, 0, 0, 0.12
0.1912046, 0.0637349]], \"target\": \"10.64.116.10\", \"sample_time\": 1650364220.061012}" 1
res = None
237 - [Topic: ms.cpu]
data_record: 2022-04-19\ 10:30:35.069738,topic=ms.cpu fields="{\"labels\": [\"time\", \"gues
\", \"irq\", \"user\", \"system\", \"nice\", \"iowait\"], \"data\": [[1650364224, 0, 0, 0.25
.1283697, 0]], \"target\": \"10.64.116.10\", \"sample_time\": 1650364225.078173}" 1
res = None
```

Figure 36 Two Netdata samples in Figure 34 which are in point (5) of Figure 28

Finally, the data in the TSDB can be retrieved from other external applications such as AE and MS. To demonstrate this, a test program was developed to query the data from the TSDB. Figure 37 shows the output of the program that gets the two sampled telemetry data; this output corresponds to point (6) in Figure 28.

```
('{"labels": ["time", "guest_nice", "guest", "steal", "softirq", "irq", "user", "system", "n
ice", "iowait"], "data": [[1650364219, 0, 0, 0.1274697, 0.3186743, 0, 4.206501, 2.676864, 0.
1912046, 0.0637349]], "target": "10.64.116.10", "sample_time": 1650364220.061012}', 'fields'
)
('{"labels": ["time", "guest_nice", "guest", "steal", "softirq", "irq", "user", "system", "n
ice", "iowait"], "data": [[1650364224, 0, 0, 0.2567394, 0.449294, 0, 5.712452, 2.439024, 0.1
283697, 0]], "target": "10.64.116.10", "sample_time": 1650364225.078173}', 'fields')
```

Figure 37 Two Netdata samples in Figure 34 which are in point (6) of Figure 28

7 MonB5G Analytics Engine for Slice-Level KPI Prediction

To support automated and proactive decisions at the slice level, the AE components provide (among other functions) predictions of slice KPIs. End-to-end KPIs at the slice level have been defined and include, for example upstream/downstream throughput for NSI, average end-to-end uplink/downlink delay, virtualized resource utilization per NSI, etc. The collection of these KPIs is supported by MS, as discussed in the previous chapter (MS). Although KPI prediction has been implemented in older systems, there are some characteristics that we need to consider when implementing it in a 5G system, at the slice level. The number of slices, as well as the amount of data collected when automatic slice redeployment is enabled, are important factors that affect the effectiveness and efficiency of AE so a scalable solution must be developed.

Our solution will include several prediction methods which are described in the following sections and are part of the AE component. Several relevant issues will be investigated through different approaches.

7.1 Local KPI Prediction

AE leverages MonB5G's distributed architecture to move analysis close to the data collection MS in each domain (i.e., RAN, Edge and Cloud), minimizing the need to transfer raw slice performance and configuration data across different network domains and slices.

bw	cpu	hdd	hhd	ram	sliceLatency
200.0	0.0	0.0	0.0	0.0	17.916667
0.0	0.0	0.0	0.0	0.0	12.068975
0.0	0.0	0.0	0.0	0.0	11.590775
159.0	0.0	0.0	0.0	0.0	17.916667
0.0	0.0	0.0	8.0	0.0	12.068975
108.0	0.0	0.0	0.0	0.0	17.916667

Figure 38 KPI inputs from MS

As shown in Figure 38 the input datasets are generated for each slice instance. These (JavaScript Object Notation (JSON)) files link the items of the State 2D array of data source files to the network elements and their function. These files are generated with the intention of being used as dataset decoder for further analysis or use by AE.

1. Dataset Files (DSs): Dataset files that contain the JSON files used by the AE engine.
2. Mapping Files (MFs): Network status mapping files that link the state of the DS files to the network elements and their function.

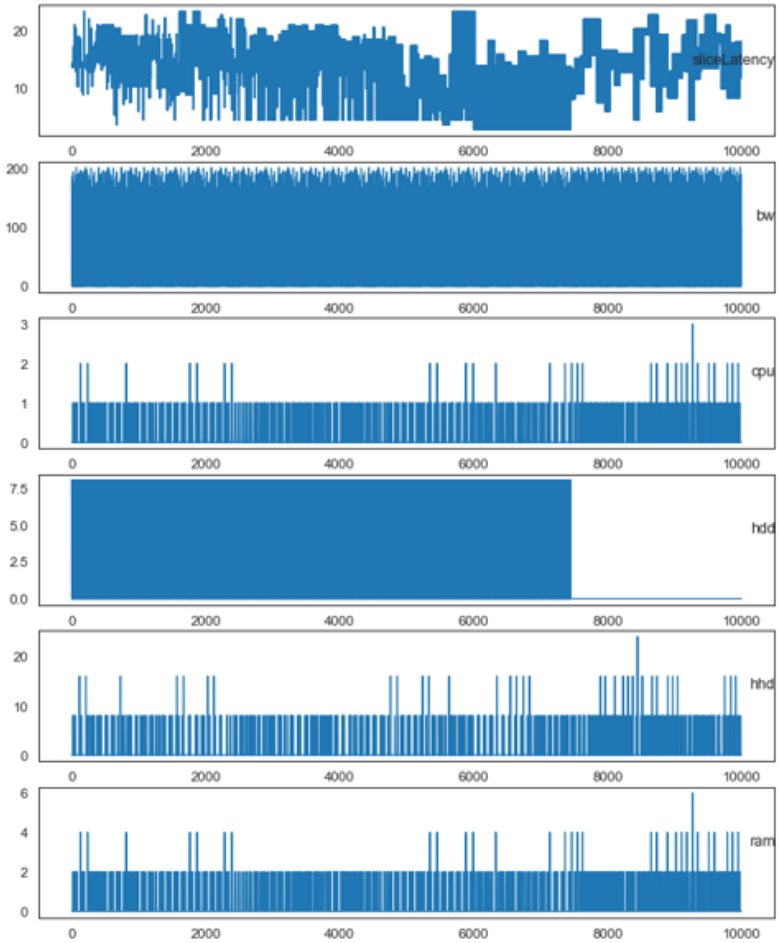


Figure 39 KPI plots from MS

Multivariate time-series analysis as shown in Figure 39 is an important statistic that simultaneously analyzes multiple measurements to study the behavior of time-dependent data, and forecasts future values depending on the history of variations in the data. We propose a GNN-based architecture to improve model predictions by incorporating network resource information (CPU, RAM, Bandwidth, and Storage) to capture the spatial dependencies between variables in the dataset.

Here, the CNN helps in extracting the feature representation. CNNs are specialized to handle data structures with multiple dimensions. In the case of 1D data, filters slide over time-series data by extracting a feature map for local subsequences in the data. They create representations for fixed-size contexts and the effective context size can easily be made larger by stacking several CNN layers on top of each other. This allows precise control over the maximum length of dependencies to be modeled. Since convolutions are common in computer graphics with direct hardware support on GPUs, CNNs are a more efficient method for extracting local patterns.

Graph Neural Networks models the relationship between a set of objects (nodes or vertices V) and their connections/interrelationships (given by a set of edges E linking the respective nodes/vertices). We define a graph as G = (V, E) where V is the set of nodes and E is the set of edges.

An edge $e_{ij} = (v_i, v_j)$ connects nodes v_i and v_j. A common way to represent and store a graph is with an Adjacency matrix $A \in \mathbb{R}^{N \times N}$ where $N = |V|$, which is a square matrix such $A_{ij} = 1$ if there is an edge from node v_i to node v_j, and 0 otherwise. The number of neighbors of a node v is known as the degree of v and is denoted by $D_{ii} = \sum_j A_{ij}$, D is then the diagonal degree matrix.

To discover hidden associations among nodes, a graph learning layer computes a graph adjacency matrix, which is later used as an input to all graph convolution modules. The graph learning layer learns a graph adjacency matrix adaptively to capture the hidden relationships among time series data. The adjacency matrix is a numerical representation of all the linkages present in the data. By propagating information through structures, graph neural networks allow each node in a graph to know the context of its neighbours. We compute the correlation matrix of slice latency between different resource KPIs, which represents our adjacent matrix. These form the node features in the network.

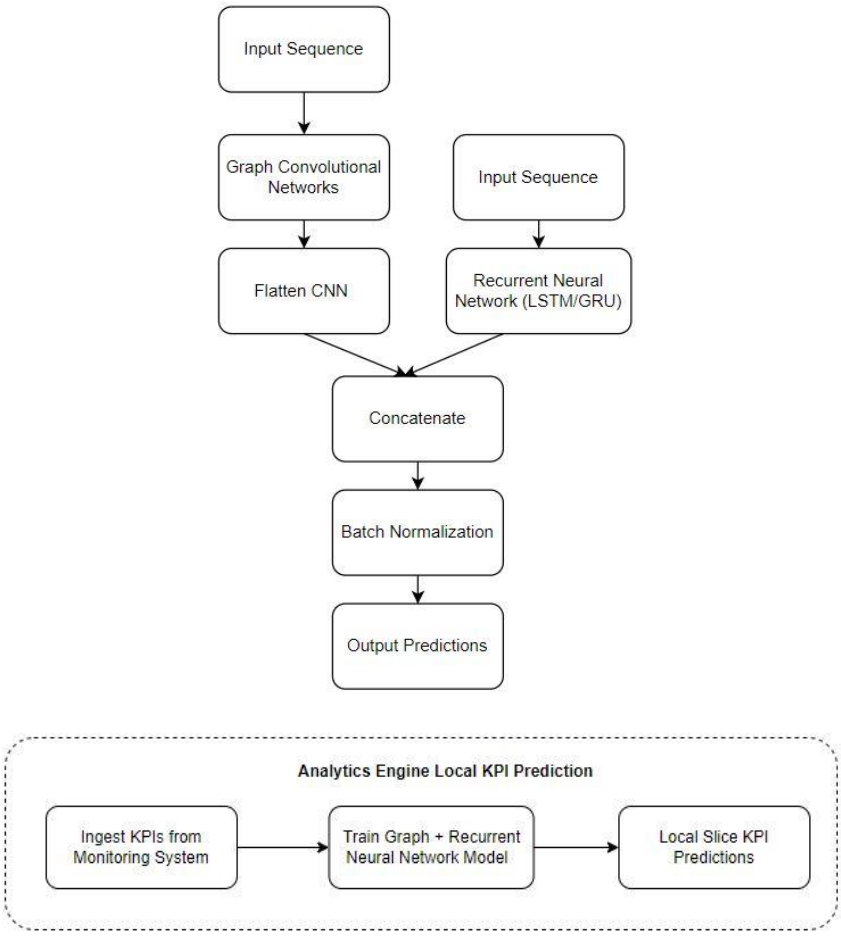


Figure 40 Training Architecture of Slice KPI Prediction Model

We propose a combination of convolution and recurrent connections that uses multiple time series as input to the network. Our model receives, as input, sequences of slice latency values and an adjacency matrix. We estimate an adjacency matrix by applying a Laplacian normalization that allows the use of an efficient layer-wise propagation rule, based on the first-order approximation of spectral convolutions. The sequences for the input model, as shown in Figure 40 are passed through the Recurrent Neural Network (LSTM/GRU) layers, while the correlation matrices are processed by Graph Convolution layers. To speed up the training, we deployed Gated Recurrent Units (GRU). We split the training and testing data as 80-20 ratio. A Batch Normalization layer was applied, which acts as a regularization and greatly improves the overall performance. The predictions are then retrieved at the end. The errors of the model are calculated as Root Mean Squared Error (RMSE) on test data. The results of the AE feature can be used for down-streaming tasks, e.g., fault management, which is discussed in more detail in Chapter 8.

7.2 Cross-Domain KPI Prediction

Dynamic resource usage forecasting for network slicing can leverage advanced Federated Learning (FL) techniques to minimise network overhead in RAN/edge/cloud domains. The traditional centralized approach to monitoring, analysing, and controlling the underlying raw data is problematic because it involves significant overhead and delays and represents a single point of failure. The decentralized approach, on the other hand, provides scalability, low data exchange and this greater security. From this perspective, distributed AI approaches, particularly FL techniques, can play a prominent role in monitoring scattered data across the network while reducing computational costs and enabling fast local analysis and decision-making.

The techniques presented in this section implement a novel scalable SLA-driven stochastic FL method for provisioning network slicing resources under SLA constraints. In particular, the contributions of the methods presented in this section are twofold:

1) To address the FL resource provisioning task in local AEs, a new approach called Statistical Federated Learning (SFL) for low overhead AE is presented. It learns the resource utilization models offline via a data distribution while respecting some predefined local SLA constraints defined in terms of long-term statistics over an observation window. The focus here is on the resource cumulative distribution function (CDF)-based SLA, which is dataset-dependent and nonconvex non-differentiable. The corresponding SFL local optimization task is formulated using the proxy-Lagrangian framework, and solved via a non-zero sum two-player game strategy.
2) **To ensure scalability in the presence of massive slicing, a novel SLA-driven stochastic FL policy is designed to select a subset of AEs that will participate in the FL task at each round**, which enhances the convergence time while maintaining the same computational cost regardless of how the number of AEs in the network increases. The proposed solution uses Docker compose and Docker containers to facilitate the development and testing of FL applications in B5G/6G networks. This subset selection policy of AEs is an enhancement of the SFL algorithm mentioned above to improve the scalability of the algorithm by selecting the number of FL agents in the learning process in a massive slicing environment.

As depicted in Figure 41, we consider a 6G topology with a Central Unit (CU)/Distributed Unit (DU) functional split where each Transmission/Reception Point (TRP) is co-located with its DU, which is connected to the corresponding CU through a fronthaul link. Each CU consists of a MS as well as an AI-enabled slice resource

analytics function, called AE. For each CU k and slice n (n = 1,…,N), MS (k, n) has a local dataset $\mathcal{D}_{k,n} = \{\mathbf{x}_{k,n}^{(i)}, y_{k,n}^{(i)}\}_{i=1}^{D_{k,n}}$ of size $D_{k,n}$ that is generally small and non-exhaustive. Therefore, the corresponding local AE participates in a federated learning task to accurately train its resource analysis and prediction model, and is connected to an end-to-end AE that resides in the Cloud domain and plays the role of a model aggregator without having access to the raw mini-datasets.

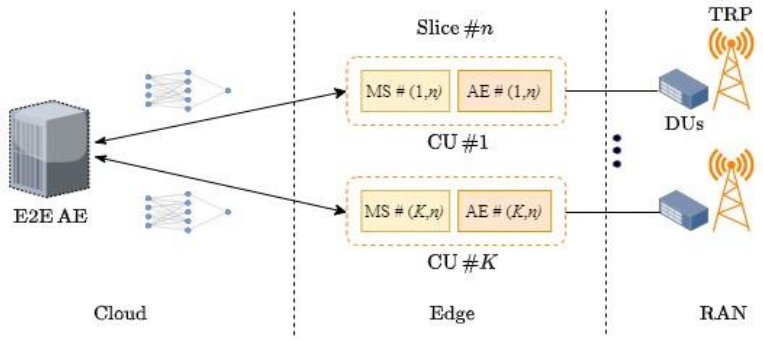


Figure 41 Network Architecture with Decentralized MS/AE at the Edge and Cloud Domains

Table 9 summarizes the input features and the supervised output of the local dataset. It includes as input features, the hourly traffic of the main over-the-top (OTT) applications, the average Channel Quality Indicator (CQI), the MIMO full-rank usage and the number of DL average active users. The supervised outputs are the number of occupied DL Physical Resource Blocks (PRBs) , the Central Processing Unit (CPU) load (in %) and the number of RRC user licenses consumed. The datasets are non-IID due to the different traffic profiles resulting from the heterogeneous distribution of users and the corresponding channel conditions. Note that this non-independent and non-identically distribution of the dataset makes it difficult to apply FL algorithms for training [47].

Table 9 Dataset Features and Output

	Metrics	Description
Features	**OTT Traffics per TRP**	Includes the hourly traffic for the top OTTs: Apple, Facebook, Facebook Messages, Facebook Video, Instagram, Netflix, HTTPS, QUIC, WhatsApp, and YouTube
	CQI	Channel quality indicator reflecting the average quality of the radio link of the TRP
	MIMO Full-Rank	Usage of MIMO full-rank spatial multiplexing in %
	# Users	Downlink average active users
Output	**CPU Load**	CPU resource consumption in %
	DL PRBs	Number of Downlink Physical Resource Blocks (PRBs) occupied
	RRC Connected Users	Number of RRC users' licenses consumed per gNB

According to the SLA established between slice n tenant and the physical operator, any assigned resources to the tenant should not exceed a range $[\alpha_n, \beta_n]$ with a probability higher than an agreed threshold γ_n. This means that the statistically constrained resource allocation model is learned under Empirical Cumulative Density Function (ECDF) and the complementary ECDF constraints, which amounts to solve the following local optimization task at FL round t (t = 0,...,T -1), i.e.,

$$\min_{\mathbf{W}_{k,n}^{(t)}} \frac{1}{D_{k,n}} \sum_{i=1}^{D_{k,n}} \ell\left(y_{k,n}^{(i)}, \hat{y}_{k,n}^{(i)}\left(\mathbf{W}_{k,n}^{(t)}, \mathbf{x}_{k,n}\right)\right),$$

$$\text{s.t. } F_{\mathbf{x}_{k,n}\sim\mathcal{D}_{k,n}}(\alpha_n) = \frac{1}{D_{k,n}} \sum_{i=1}^{D_{k,n}} \mathbb{1}\left(\hat{y}_{k,n}^{(i)} < \alpha_n\right) \leq \gamma_n,$$

Empirical CDF

$$\tilde{F}_{\mathbf{x}_{k,n}\sim\mathcal{D}_{k,n}}(\beta_n) = \frac{1}{D_{k,n}} \sum_{i=1}^{D_{k,n}} \mathbb{1}\left(\hat{y}_{k,n}^{(i)} > \beta_n\right) \leq \gamma_n,$$

where ℓ(.) is the squared error loss function, 1(.) represents the indicator function.

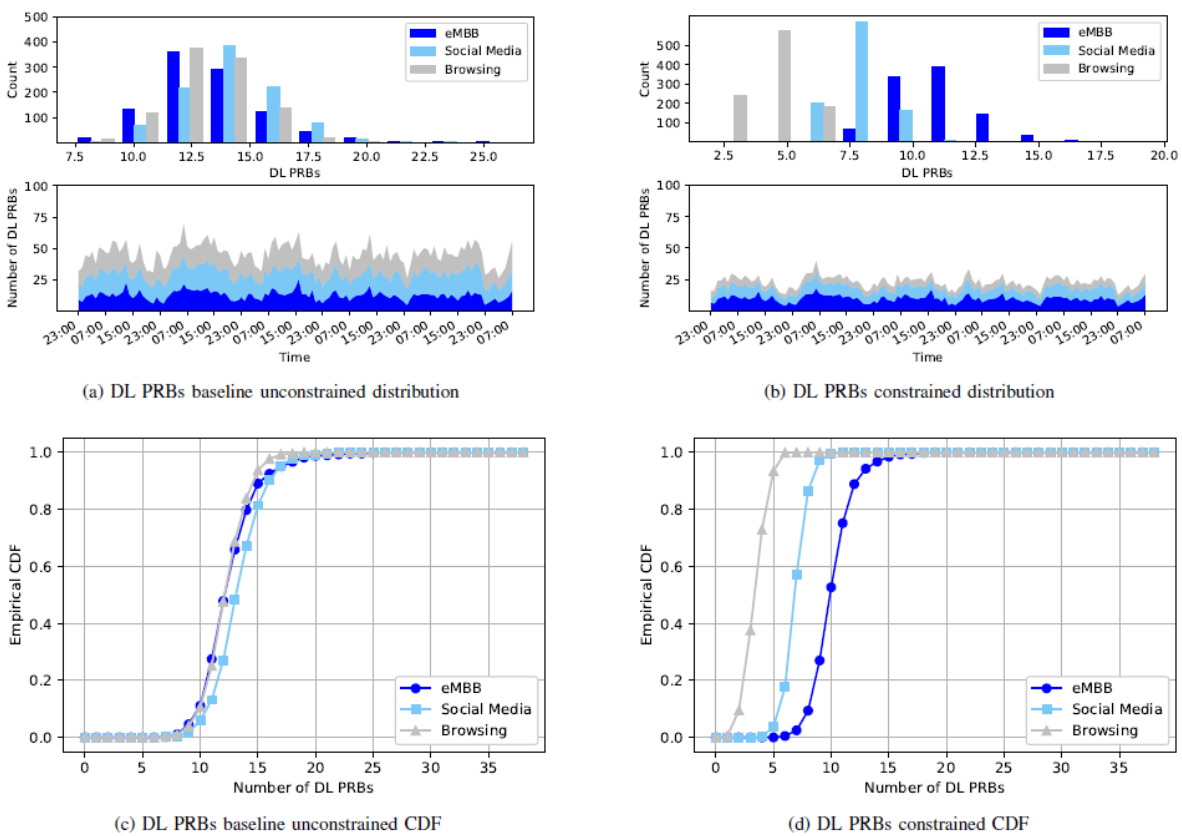
The local Statistical Federated Learning (SFL) optimization can be solved using a proxy-Lagrangian approach [48] that consists of forming two Lagrangians. The first one, $\mathscr{L}_1$, contains the loss function and a smooth approximation to the SLA constraints, called proxy constraints, where the indicators are replaced by smooth sigmoid functions. The second Lagrangian, $\mathscr{L}_2$, consists of the original SLA constraints. The joint optimization of the two Lagrangians turns out to be a non-zero-sum two-player game, where the first player wants to minimize $\mathscr{L}_1$ and the second player wants to maximize $\mathscr{L}_2$. This process leads to a near-optimal and near-feasible solution (i.e., all constraints are nearly satisfied) to the original constrained problem. The obtained weights are then sent back to the FL server to perform the aggregation of the local models. This process is summarized in Algorithm 1. The details of the two-player game are omitted from this section, interested readers are referred to reference [49].

Algorithm 1: Federated learning with local proxy-Lagrangian two-player game for slice n.

Input: R_λ, η_λ, T, L.
OSS server initializes $\mathbf{W}_n^{(0)}$ with random Gaussian weights and broadcasts it to local AEs.
for $t = 0, \ldots, T-1$ **do**
  parallel for $k = 1, \ldots, K$ **do**
  Initialize $M =$ `num_constraints` and $\mathbf{W}_{k,n,0} = W_n^{(t)}$
  Initialize $\mathbf{A}^{(0)} \in \mathbb{R}^{(M+1)\times(M+1)}$ with $\mathbf{A}_{m',m}^{(0)} = 1/(M+1)$
  for $l = 0, \ldots, L-1$ **do**
    Let $\lambda^{(l)}$ be the top eigenvector of $A^{(l)}$
    `# Oracle optimization`
    Let $\mathbf{W}_{k,n,l} = \mathcal{O}_\delta\left(\mathcal{L}_{\mathbf{W}_{k,n,l}}(\cdot, \lambda^{(l)})\right)$
    Let $\Delta_\lambda^{(l)}$ be a gradient of $\mathcal{L}_\lambda(\mathbf{W}_{k,n,l}, \lambda^{(l)})$ w.r.t. λ
    `# Exponentiated gradient ascent`
    Update $\tilde{\mathbf{A}}^{(l+1)} = \mathbf{A}^{(l)} \odot \cdot \exp\left\{\eta_\lambda \Delta_\lambda^{(l)}(\lambda^{(l)})\right\}$
    `# Colunm-wise normalization`
    $\mathbf{A}_m^{(l+1)} = \tilde{\mathbf{A}}_m^{(l+1)} / \left\|\mathbf{A}_m^{(l+1)}\right\|_1$, $m = 1, \ldots, M+1$
  end
  return $\mathbf{W}_{k,n}^{(t)} = \mathbf{W}_{k,n,L-1}$
  Each local AE (k,n) sends $\mathbf{W}_{k,n}^{(t)}$ to the OSS server.
  end parallel for
  return $\mathbf{W}_n^{(t+1)} = \sum_{k=1}^{K} \frac{D_{k,n}}{D_n} \mathbf{W}_{k,n}^{(t)}$
  and broadcasts the value to all local AEs.
end

To evaluate the performance of the proposed statistical FL technique, three primary slices (eMBB, social media, and browsing) are considered next to analyze the proposed FL policy. The traffic corresponding to each slice is the sum of the associated OTT traffic collected from hourly traffic over five days. The slices are as follows: **1) eMBB**: Netflix, YouTube and Facebook video; **2) Social Media**: Facebook, Facebook Messages, WhatsApp and Instagram; **3) Browsing**: Apple, http and QUIC.

The PRB resources are dynamically allocated to the slices according to the corresponding traffic patterns and radio channel conditions (i.e., MIMO full-rank usage and CQI), as shown in Figure 42 (a) and (b), while ensuring the long-term isolation of slices through the constraints on the CDF of the underlying physical resources. A total number of *K=200* AEs was simulated. We use $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ vectors to denote the upper and the lower assigned resource bounds, respectively, to each the three slice types (i.e., eMBB, social media and browsing), $\boldsymbol{\alpha}$ = [0,0,0], $\boldsymbol{\beta}$ = [15, 10, 10] PRBs and $\boldsymbol{\gamma}$ = [0.01, 0.01, 0.01] denote the probability threshold for different slices. When no constraints are enforced, as shown in Figure 42 (c), all three slices violate their upper bounds with a high probability, which can be considered unacceptable by both operators and slice tenants. Figure 42 (d) shows that the number of allocated PRBs is within the corresponding upper and lower bounds $[\alpha_n, \beta_n]$ with a 99% probability.



(a) DL PRBs baseline unconstrained distribution

(b) DL PRBs constrained distribution

(c) DL PRBs baseline unconstrained CDF

(d) DL PRBs constrained CDF

Figure 42 DL PRBs Distributions, with α = [0, 0,0], β = [15, 10, 10] PRBs and γ = [0.01, 0.01, 0.01].

To ensure **scalability under massive slicing**, a new SLA-driven stochastic FL policy is next presented to select the subset of AEs that participate in the FL task in each round. This approach will enhance the convergence time, as the AEs that have smaller number of SLA violations will have a greater chance of participating the FL task. Since the cardinality of the subset of selected AEs remains fixed, the computational cost of each FL round remains the same regardless of how the total number of AEs increases.

To ensure scalability in a massive slicing scenario, we aim to select a subset of active AEs in each round of the federated learning process to optimize the FL computation time and the underlying resource computation. To this end, a new SLA-driven stochastic policy is considered for selecting the AEs. Once the training round is completed at round *t*, each AE *(k,n)* evaluates the generalization of its FL model using a test dataset $\widetilde{\mathcal{D}_n}$ of size $\widetilde{D}_n$, which is common to all monitoring systems of slice *n*, and computes the SLA violation rate over the common test dataset.

Next, at each FL round *t*, all the AEs send their violation rates derived from this test dataset, denoted as $\nu_{k,n}$, to the FL aggregation server, which generates a probability distribution $\pi_{k,n}$ using a *softmin* function as

$$\pi_{k,n} = \frac{\exp\{-\nu_{k,n}\}}{\sum_{p=1}^{K} \exp\{-\nu_{p,n}\}}.$$

The *softmin* function rescales the values so that the elements are in the range [0, 1] and the sum of all elements is equal to 1. Based on the above probability distribution, only a subset of *m<K* AEs are considered at each FL round to participate in the learning process. In other words, only a subset of AEs would stochastically participate in the FL task. The IDs of the *m* selected AE agents are generated from a non-uniform random sample where the probabilities for each AE depend on $\pi_{k,n}$. Under this stochastic policy, AEs with low SLA violation rates receive a high probability of participation in the FL round to drive convergence of the model, but AEs with high SLA violations could also participate in FL training with a low probability to guarantee the generalization that could emerge from their datasets.

The model averaging in the server at round *t* is calculated as

$$\mathbf{W_n^{(t+1)}} = \sum_{k \in \{k_1,\ldots,k_m\}} \frac{D_{k,n}}{D_n} \mathbf{W_{k,n}^{(t)}}$$

where D_n, in this case, denotes the sum of the sizes of the datasets that cooperate in the FL task. The proposed method is summarized in Algorithm 2 [50].

Algorithm 2: SLA-Driven Stochastic Federated Learning Policy.

Input: K, m, η_λ, T, L.
parallel for $k = 1, \ldots, K$ **do**
Calculate SLA based violation rate
AE (k, n) calculates $\nu_{k,n}$ and reports it to the aggregation server
end parallel for
# Federated Learning
# Server generates probability distribution using Softmin function
for $k = 1, \ldots, K$ **do**
  $\pi_{k,n} = \frac{\exp\{-\nu_{k,n}\}}{\sum_{l=1}^{K} \exp\{-\nu_{l,n}\}}$, $k = 1, \ldots, K$
end
Server initializes $\mathbf{W}_n^{(0)}$ with initial training parameter
for $t = 0, \ldots, T-1$ **do**
  # Server selects the m AEs ID using np.random.choice
  $\mathrm{AE}_{k_1,n}^{(t)}, \ldots, \mathrm{AE}_{k_m,n}^{(t)} \sim \{\pi_{1,n}, \ldots, \pi_{K,n} \mid \mathrm{AE}_{1,n}, \ldots, \mathrm{AE}_{K,n}\}$
  Server broadcasts $\mathbf{W}^{(0)}$ to the m selected AEs
  parallel for $k \in \{k_1, \ldots, k_m\}$ **do**
  # Local epochs
  for $l = 0, \ldots, L-1$ **do**
    Solve the proxy-Lagrangian game between $\mathcal{L}_{\mathbf{W}_{k,n}^{(t)}}$ and $\mathcal{L}_\lambda$ and get $\mathbf{W}_{k,l}$
  end
  return $\mathbf{W}_{k,n}^{(t)} = \mathbf{W}_{k,L-1}$
  Each local AE k sends $\mathbf{W}_{k,n}^{(t)}$ to the aggregation server.
  end parallel for
  # FL Server Aggregation
  return $\mathbf{W}_n^{(t+1)} = \sum_{k \in \{k_1, \ldots, k_m\}} \frac{D_{k,n}}{D_n} \mathbf{W}_{k,n}^{(t)}$
  Broadcasts $\mathbf{W}_n^{(t+1)}$ to all K AEs.
end

To evaluate the feasibility of the proposed technique in a real scenario and to prove the scalability of the proposed policy, a cloud native approach of the FL agents was implemented. To emulate a cloud-native deployment, the Docker compose tool was considered. The main reason for choosing Docker compose is that it usually runs on Kubernetes. The FL server and the agents (i.e., AEs) communicate with each other via REST API. Since this section focuses on the algorithm overview, the details of this cloud-native implementation are omitted here. We refer interested readers to reference [50].

In the following, three primary slices (eMBB, social media and browsing) are considered to evaluate the proposed FL policy. The traffic corresponding to each slice is the sum of the associated OTT traffic, collected from hourly traffic over five days. These three slices are split as follows: 1) eMBB: Netflix, YouTube, and Facebook video; 2) Social Media: Facebook, Facebook Messages, WhatsApp and Instagram; 3) Browsing: Apple, http, and QUIC. The parameter settings are presented in Figure 43.

Parameter	Description	Value
N	# Slices	3
K	# AEs	100 (Simul.), (50, 40) (Emul.)
m	# Selected AEs	50 (Simul.), 25 (Emul.)
$D_{k,n}$	Local dataset size	1000 samples
T	# Max FL rounds	30
L	# Local epochs	160
R_λ	Lagrange multiplier radius	Constrained: 10^{-5}
η_λ	Learning rate	0.02

Figure 43 Parameter settings of the scalability test

To illustrate the **faster convergence** of the proposed technique, the next figure shows that the policy-based FL converges faster than the FL without policy because the selected AEs (i.e., FL agents) with the lower violation rate have a higher probability of participating in the training. In this case, a total number of *K*=100 AEs were simulated, and *m*=50 AEs were selected for the FL task. We use $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ vector to denote the upper and the lower assigned resource bounds, respectively, to each the three slice types (i.e., eMBB, social media and browsing), $\boldsymbol{\alpha} = [0,0,0]$, $\boldsymbol{\beta} = [4,7, 10]$ % and $\boldsymbol{\gamma} = [0.01, 0.01, 0,01]$ denote the probability threshold for different slices. The proposed subset selection policy is able to provide faster convergence than the FL with all possible AEs.

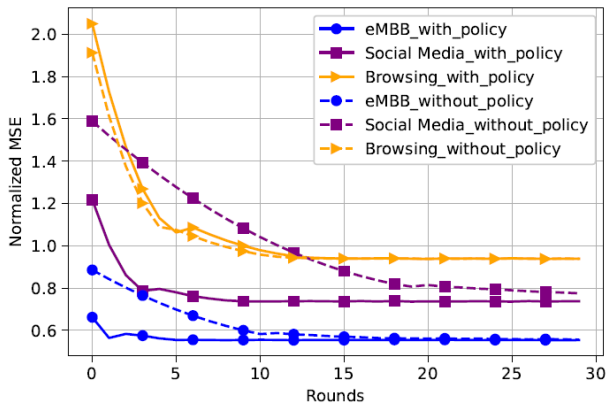


Figure 44 MSE loss as a function of the number of FL rounds with and without policy (simulated scenario).

Figure 45 shows the results when the proposed algorithm was trained with *K* = [40, 50] Analytic Engines. In both cases, the proposed method converges faster when only *m* = 25 AEs are selected for cooperation in the FL task for the emulated (containerized) solution. The values of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are those presented in the previous paragraph.

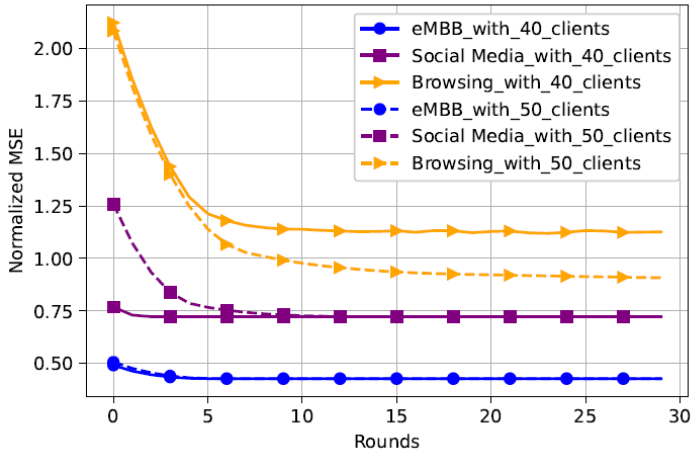


Figure 45 MSE loss as a function of the number of FL rounds (emulated/containerized scenario)

To show the **scalability** of the proposed solution, Figure 46 presents the computation time versus the rounds for the simulated and the emulated (containerized) system. Note that the computational load depends on the size of

the physical computer. Interestingly, since we set the cardinality of the subset of AEs to *m* = 25 AEs, the computation cost remains at the same level regardless of the total number of AEs *K* = [40, 50].

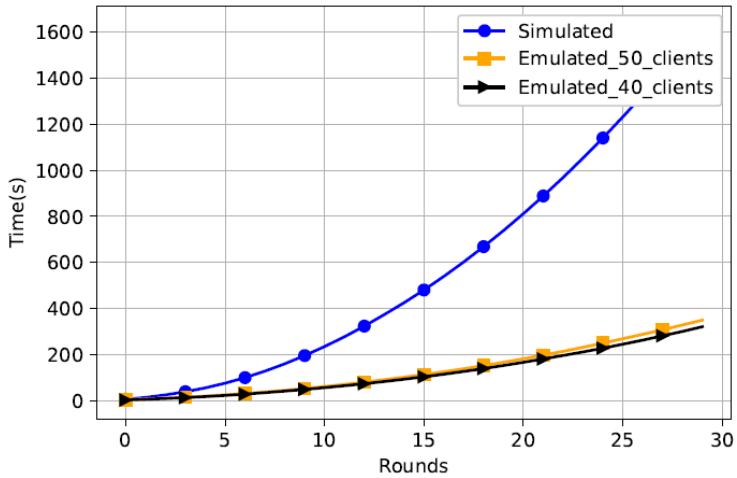


Figure 46 Convergence time of simulated vs emulated (containerized) solution with the proposed policy (m=25, K=[40,50])

Finally, the **violation rate** observed in the policy-driven constrained FL presented in this section shows a significantly lower number of SLA violations compared to the traditional FedAvg approach, which has violation rates of 0.275, 0,124 and 0,923 for eMBB, social media and browsing slices, respectively, while maintaining the performance of the constrained FL technique without policy, (i.e., around 1%).

As shown in deliverable D3.1 [3] and [49], SFL can reduce the communication overhead more than ten times compared to a centralized constrained learning baseline. The selection policy presented in this section further reduces this overhead by limiting the number of agents that cooperate during the FL task, which promotes scalability in a massive network slicing scenario.

7.3 Network Aware KPI Prediction

7.3.1 CONTEXT-AWARE DEMAND PREDICTORS

Instead of *accurately* predicting traffic, it is also possible to develop predictors consider knowledge related to network slicing [51] and resource orchestration [52], making them in fact *context-aware*. The context in this case is implicitly define as the resource orchestration problem in 5G (and Beyond-5G, for that matter) networks for network slicing. Even though the type of resource demand and resource prediction is agnostic to this type of resource demand predictor, for this project the consortium developed a resource demand predictor that focuses on predicting traffic load.

The knowledge that makes the predictor context-aware is formulated as constraints that describe the relation between resource prediction and resource allocation: underestimating slice resource load prompts orchestrators to under-provision slice resources, which causes (among other things) SLA violations; and overestimating res yields resource over-provisioning with penalties of its own, but it can be desirable to some

degree given the fluctuation-prone behavior of traffic [53]. This knowledge is included as regularization terms in the loss functions used to train the DNNs [54]. This predictor we called it the "Context-Aware Traffic Predictor", CATP for short.

In order to enhance the capability of traffic predictors and reduce the gap between the prediction of resource demand and resource orchestration, it is necessary for the predictors to include problem domain knowledge. When solving regression problems with DNN function approximators, it is common to use the Mean Squared Error (MSE) or Mean Absolute Error (MAE) as the loss functions, the latter shown in Eq.1 in which B is the batch of values used in the current iteration, y_i is the ground-truth values of the variable (i.e. real load in the context of this paper) and y_i^p is the predicted value (predicted load). In a resource orchestration setting, the predicted value of the load y_i^p is used to drive the resource orchestration policies.

$$MAE = \frac{\sum_{i=1}^{B} |y_i - y_i^p|}{B} \qquad (Eq.1)$$

Using MAE as the basis loss function, we can define a cost model for the MAE function according to Eq. 2

$$MAE_{cost} = |y_i - y_i^p| \qquad (Eq.2)$$

The capability of a predictor can be enhanced by including regularization terms to the MAE cost model, resulting in a new cost model expressed in Eq. 3.

$$C_{model} = |y_i - y_i^p| + \lambda_h * H_{reg}(y_i, y_i^p) \qquad (Eq.3)$$

The second term of Eq. 3, H_{reg}, embeds problem domain knowledge related to resource orchestration in 5G/B5G networks. This term can be viewed as a *constraint* for the predictor. The λ is a weighting factor which, as it becomes larger, will drive the predictor into satisfying the resource orchestration constraints given by H_{reg}. But as λ becomes smaller, then the loss function prioritizes prediction accuracy.

In the context of 5G/B5G networks, the terms y_i and y_i^p of Eq. 3 can represent the real load (i.e. real resource demand) and the predicted load (i.e. the predicted resource demand), and the latter can be treated as the equivalent to the *resources allocated* to the element handling the load. Thus, Eq. 2 can also be interpreted as the difference between the real resource demand and the resource allocation (based on the predicted load). By considering this, the function H_{reg} in Eq. 3 can be defined as shown in Eq. 4.

$$H_{reg}(y_i, y_i^p) = \begin{cases} H_{u-p}(y_i, y_i^p) & y_i > y_i^p \\ H_I(y_i, y_i^p) & y_i \approx y_i^p \\ H_{o-p}(y_i, y_i^p) & y_i < y_i^p \end{cases} \qquad (Eq.4)$$

Equation 4 defines a piece-wise function $H_{reg}(y_i, y_i^p)$ as the regularization term. In Eq. 4 (and here on forward), the subscripts *u-p*, *I* and *o-p* stand for *under-provisioning*, *ideal* and *over-provisioning*, respectively. If the conditionals in this function are dependent on the domain of y_i and y_i^p, with associated functions $H_*(y_i, y_i^p)$, then $H_{reg}(y_i, y_i^p)$ becomes an approximation constrained (regularization) function [55].

When a network slice is under-provisioned of resources due to predictor estimating a resource demand smaller than the real resource demand ($y_i > y_i^p$), the following sources of penalties appear:

- A penalty proportional to the difference between the amount of resources given to the slice (corresponding to the predicted load y_i^p) and the real load of the slice. Assigning less resources based on the predicted load y_i^p will increase the probability of an SLA violation
- A penalty associated to re-arranging the resource allocation to meet the demand of an under-provisioned network slice. Re-assigning resources requires hardware and software support, which increases the total cost of ownership for the infrastructure provider (InP) [56].

The cost of under-predicting resource demand, which results in resource under-provisioning can be represented as growing linearly as a function of the difference between the predicted and real resource demand. This is represented in the first term after the equality in Equation 5 with the parameter C_d. This latter represents a gain parameter for the under-provisioning cost. As C_d grows, the predictor will be trained with a high-penalty of resource under-provisioning, learning to avoid this condition as much as possible.

$$H_{u-p}(y_i, y_i^p) = (C_d + C_r) * (y_i - y_i^p). \quad (Eq.5)$$

The second term after the equality in Equation 5 represent the cost re-arranging resources. Such an operation will occur if some resource unde-provisioning occurs in order to prevent an increase in the probability of SLA violations. Similar to C_d the parameter C_r represents a gain for the re-arranging costs, and as C_r grows, the predictor will be trained with a high-penalty of resource re-arrangement. In most cases, it is expected that $C_d \gg C_r$, since the operational penalty of over-provisioning is usually much larger and carries a lot of negative implications for the service provided and the infrastructure owner, in terms of lost revenue and associated losses.

When a network slice is over-provisioned of resources due to predictor over-estimating the resource demand with respect to real resource demand ($y_i < y_i^p$), the different sources of penalties are:

- A penalty related to the idle resources, which reduces revenue potential for the InP.
- An operational cost incurred by the InP due to re-configuring resource allocation
- Assuming full resource allocation and resource *overbooking*, network slices with idle resources increases the chances of under-provisioning of other network slices, and preventing more slices and their respective user service requests to be admitted for execution.

These three sources of penalties will generate Equation 6, which has some resemblance with Equation 5, since both consider different a growing linear behavior (linear) for the penalties associated to any type of under-provisioning/under-estimation. The costs associated to over-provisioning and resource re-arrangement are both consider linear with respect to the difference of the estimated (predicted) resource demand and the real resource demand.

$$H_{o-p}(y_i, y_i^p) = (C_w + C_r + P_r C_d) * (y_i^p - y_i). \quad (Eq.6)$$

In Equation 6, the parameter C_w is a gain that assigns a numerical penalty to the over-provisioning of resources, which are in fact wasted resources. The parameter C_d has the same meaning as it was the case of the penalty associated with under-provisioning (under-estimation of resource demand), while the P_r represents the sensitivity of the system model to the condition of indirect under-provisioning of other network slices due to over-provisioning of a particular slice.

Using both sets of equations outlined above, it is possible to obtain a set of loss functions that can be used to train a neural network to work as a time series predictor. Two variations result from varying the conditionals of the regularization function. A new parameter is added, y_s, called "slack" that allows to customize the region of ideal allocation depending on system behavior. Equation 7 forces y_s to be above zero, which imposes a bias on the predictor against under-provisioning (thus, under-estimation) of real resource demand. Equation 8 creates some flexibility, by allowing the ideal allocation region to revolve around the situation in which $y_i = y_i^p$, allowing for the slack y_s to be spread evenly below and above the case in which $y_i = y_i^p$.

$$L_{catp01}(y_i, y_i^p) = \begin{cases} (C_d + C_r) * (-\Delta x) & \Delta x < 0 \\ 0 & 0 \leq \Delta x \leq y_s \\ (C_w + C_r + P_r C_d) * (\Delta x - y_s) & \Delta x > y_s \end{cases} \quad (Eq.7)$$

$$L_{catp02}(y_i, y_i^p) = \begin{cases} (C_d + C_r) * \left(\frac{y_s}{2} - \Delta x\right) & \Delta x < -\frac{y_s}{2} \\ 0 & |\Delta x| \leq \frac{y_s}{2} \\ (C_w + C_r + P_r C_d) * \left(\Delta x - \frac{y_s}{2}\right) & \Delta x > \frac{y_s}{2} \end{cases} \quad (Eq.8)$$

The evaluation of CATP consisted in training state-of-the-art DNNs with ECATP formulated loss functions. The implementation is in Python 3.8 and Tensorflow 2.1. Table 10 describes the DNN architectures and the parameters used for each. The hyperparameters chosen for the STN [57] and the 3D-CNN [58] architectures were chosen based on the recommendations by their respective authors. STN stands for "Spatio-Temporal Neural Network" and it uses an encoder-decoder paradigm, combining a stack of Convolutional Long Short-Term Memory (ConvLSTM) and three-dimensional Convolutional Network (3D-ConvNet) elements. In the case of LSTM and Conv-LSTM2D, the hyper-parameters were chosen based on empirical experimentation, choosing the best performing ones for a trial of experiments using a portion of the available dataset.

Table 10 DNNs evaluated with CATP and their parameters. All DNNs use a learning rate of 0.001 with Adam Optimizer, and 100 Epochs.

DNN	Parameters						Description
	Hidden Layers	Hidden Layer/ Filter Size	Activation Function	Dropout Rat	Batch Size	Unrolling	
LSTM	4	50	Tanh	0.2	42	24	Widely used for time-series prediction
3DCNN	4	32-16-64-32	ReLu	0.3	128	24	DNN architecture in [58]
STN	9	3-3-3-6-6-6-6-6-4	ReLu	-	128	6	DNN architecture in [57]

Conv-LSTM2D	4	4	tanh	-	128	6	Using Conv-LSTM2D (CL2D) as hidden layers

Different loss functions were tested with each DNN. Our baseline for comparison consists of three loss functions: MAE, MSE and the loss function in [58]. We implemented the two loss functions L_{catp01} and L_{catp02} previously outlined. Every DNN architecture was paired with these loss functions, generating 8 different (DNN, Loss Function) pairs. For every pair, a large set of experiments were run with different values and combinations of the parameters C_w, C_d and C_r, resulting in a larger number ($\sim 10^2$) of experiments. All the DNNs were trained using mobile traffic data from the city of Milano [59]. However, our predictors are agnostic to this and it is possible to train them using any time-series data from any generation of technology and/or service type.

In order to evaluate the effectiveness of each (DNN, Loss Function) pair, we introduce a total penalty cost function TC (Equation 9) that quantifies the resource demand mispredictions of a (DNN, Loss Function), with different cost of misprediction for under- and over-provisioning cases. TC can be interpreted as the monetary or operational cost of resource demand of misprediction. In its formulation, TC uses the ratio between the cost of over-provisioning to the cost of under-provisioning, the latter of which is usually larger. The expression for TC is given by Equation 9 (which in practice tends to be smaller) over the real cost of under-provisioning, and expresses it as in Equation 10 across a series of predictions and ground-truth values.

$$TC = \sum_{i=1}^{T} P_i(y_i, y_i^p) \qquad (Eq.9)$$

$$P_i = \begin{cases} CR_{\frac{OP}{UP}} * (y_i^p - y_i) & \text{if } y_i^p > y_i \\ 1.0 * (y_i - y_i^p) & if\ y_i > y_i^p \end{cases} \qquad (Eq.10)$$

This ratio allows to understand TC as a function of under-provisioning, which tends to be more significant in real case scenarios. For example, for $CR_{\frac{OP}{UP}} = 0.5$, the resulting TC will correspond to the case in which the impact of over-estimating demand (resulting in resource over-provisioning) is 50% that of the impact of under-estimating demand (under-provisioning).

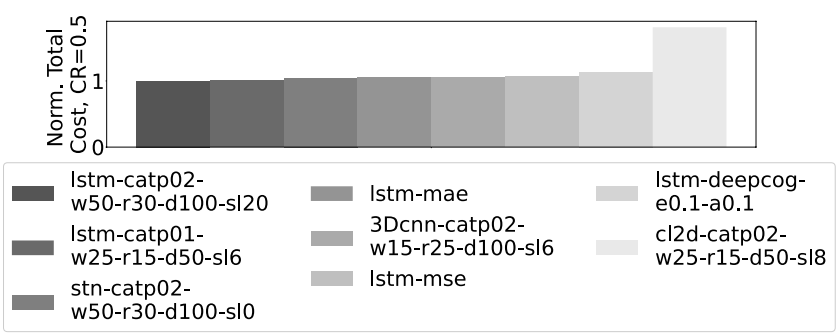


Figure 47 Normalized Total Penalty for $CR_{\frac{OP}{UP}} = 0.5$ pairs (less is better)

Figure 47 shows the normalized penalty for $CR_{\frac{OP}{UP}} = 0.5$. The (LSTM, L_{catp02}) pair performs the best with $C_w = 50.0$, $C_r = 30.0$, $C_d = 100.0$ and $y_s = 20.0$. Following very closely, it is the (LSTM, L_{catp01}) pair with $C_w = 25.0$, $C_r = 15.0$, $C_d = 50.0$ and $y_s = 6.0$, which has a penalty 1.1% higher. Most of the pairs for this case have similar performance, being the (CL2D, L_{catp02}) the least performing one, which is 81.1% higher than (LSTM, L_{catp02}), while the (LSTM, $L_{deepcog}$) is 13.5% higher than (LSTM, L_{catp02}).

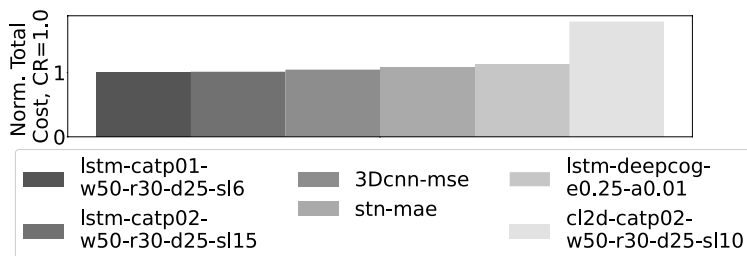


Figure 48 Normalized penalty for $CR_{\frac{OP}{UP}} = 1.0$.

Figure 48 shows the normalized penalty for $CR_{\frac{OP}{UP}} = 1.0$. In this case, it is the (LSTM, L_{catp01}) pair with $C_w = 50.0$, $C_r = 30.0$, $C_d = 25.0$ and $y_s = 7.0$ that performs the best. Closely following, it is the (LSTM, L_{catp02}) pair with a penalty 1.3% larger (worse) using $C_w = 50.0$, $C_r = 30.0$, $C_d = 25.0$ and $y_s = 15.0$. The (CL2D, L_{catp02}) pair performs the least with a penalty 79.4% higher than the first one, while the pair (LSTM, $L_{deepcog}$) with $\epsilon = 0.25$ and $\alpha = 0.01$ has a penalty 13.3% higher.

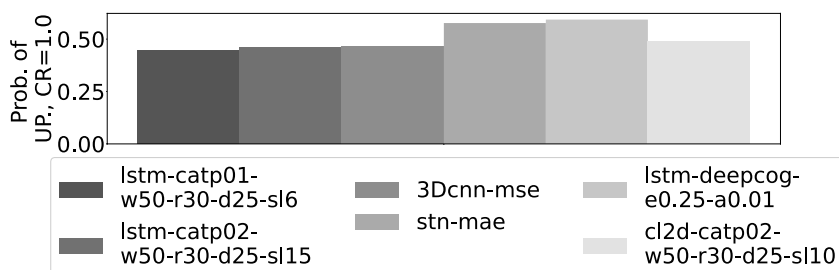


Figure 49 Probability of under-provisioning for $CR_{\frac{OP}{UP}} = 1.0$ (less is better).

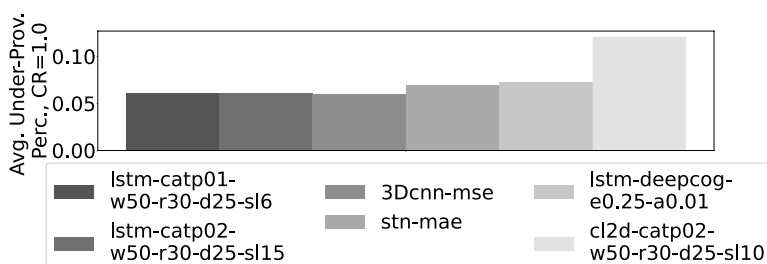


Figure 50 Average under-provisioning percentage for $CR_{\frac{OP}{UP}} = 1.0$ (less is better).

Figure 49 and Figure 50 show the probability of SLA violation (i.e. probability of under-provisioning) and the average magnitude of under-provisioning, respectively. The ones performing the best have a lower average percentage and probability of under-provisioning, while the least performing pairs, (LSTM, $L_{deepcog}$) and (CL2D, L_{catp02}), have the highest values.

The results show that CATP-derived loss functions achieve the best results consistently. The problem domain knowledge formulation with the approximation constraints generate prediction values that yield the least penalty with respect to resource allocation, reducing the gap between resource demand prediction and orchestration.

7.3.2 *RAN-MEC SPLIT CONVNET ARCHITECTURES FOR NETWORK-AWARE KPI PREDICTION*

As explained from the beginning of the project, sophisticated architectures based on deep neural networks have been heavily explored recently for various wireless network related prediction tasks. While these approaches are promising, the standard assumption of a *centralized* implementation of the DNN architectures raise significant challenges when used to control key 5G+ network functions. *First*, unlike the use of DNNs for application layer tasks (e.g., image classification on a phone) that can be *lazily* offloaded to a central computational cloud, the use of DNNs for controlling key 5G network functions (e.g., allocation of RAN resource blocks among tenants and/or users) require significantly lower latency, which might not be satisfiable by sending all required data to a central DNN, making the decision there, then sending it back the actuation message to the desired edge components; *Second*, constantly sending raw monitored data over a possibly already congested network towards a DNN architecture lying deep in the core might have a prohibitive network footprint. As a result, in MonB5G we have been exploring how such DNN-based architectures could be appropriately *implemented in a distributed fashion,* both for KPI prediction as well as reacting to such predictions in the decision engine, e.g., resource scaling (to be discussed further in the context of WP4 deliverables). The main goal of this thread has been resolving both of the above concerns *towards greatly increased scalability, yet without compromising the observed accuracy advantages of a larger (centralized) DNN architecture*.

In our initial exploration of the topic in D3.1 [3], we discussed how a DNN architecture based on 3D convolutional neural networks [58] can be effectively split between:

- An edge DNN (e.g., running at the RAN or edge cloud) of the DNN that is lightweight, i.e., shallower than a baseline centralized DNN: this edge DNN can make sufficiently accurate KPI predictions *locally,* in the majority of cases, thus (i) greatly increasing prediction latency (due to both fewer DNN layers in the forward pass, as well as proximity to the monitored signals); (ii) alleviating the network load to collect raw monitored data to a centralized DNN and thus improving the architectures scalability; (iii) allowing in many instances the (also) local DE component to take immediate decisions, compared to a more rigid centralized AE/DE.
- A central DNN (running at a core or central cloud location) that is significantly more powerful (and thus slower) and is tasked with improving the accuracy of local KPI predictions of the edge DNN, when the latter are not sufficiently confident.

- An offloading component that must frugally decide, at runtime, which predictions to delegate to the central DNN, and how to best optimally combine local and central decisions, when the latter is needed.

Figure 51 describes such an example D(istributed) DNN architecture that can be used for either traffic prediction of potentially correlated sources (e.g., the future traffic demand of 4 base stations or 4 VNFs belonging to the same slice) or also take direct action, e.g., reallocating resources to satisfy an SLA related to these demands (to be further detailed in WP4).

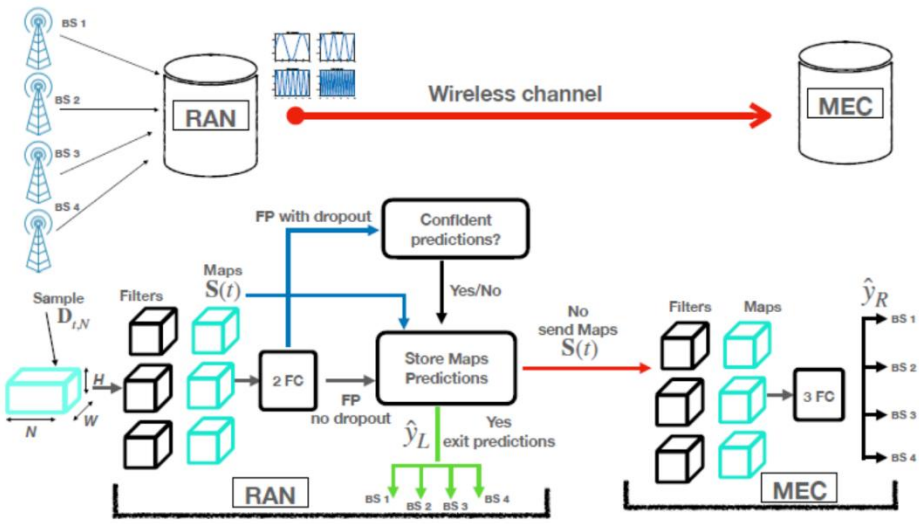


Figure 51 DDNN Architecture for Network-Aware KPI Prediction: the Local Part of the NN in the RAN (near the BSs) and the Remote Part in the MEC

Progress since D3.1. In the deliverable D3.1 [3], the distributed architecture was proposed to apply in a single domain only: the traffic demand of up to 16 RAN components (base stations in the experiments considered) was predicted at the local edge component, consisting of 1 convolutional layer and 2 fully connected ones, and supplemented by a centralized component; a heuristic mechanism was used to decide whether the local traffic prediction was confident enough, or else the output from the local convolutional layer had to be transmitted to a centralized component running 2 additional convolutional layers and 3 fully connected ones. Since this initial investigation, the work has greatly progressed along a number of key directions.

- *Cross-domain:* The distributed architecture can now support slices that contain components across different technological domains. For example, each slice can consist of different VNFs and links connecting those, with potential correlations in the demand of involved components being leveraged during prediction.
- *Composite KPIs:* While during the first phase we considered simple MSE for prediction (as well as symmetric objectives/SLAs for allocation problems in WP4), the architecture can now predict sophisticated KPIs that depend on the performance of all components (VNFs, links, across domains) of a slice. Examples include end-to-end delay through a VNF chain or bottleneck bandwidth along a VNF path.

- *Optimized prediction offloading mechanism*: Perhaps, the most important improvement of the architecture has been the *optimization of the network-aware offloading decision* at the edge. Specifically, a 3rd neural network is trained to predict whether the additional accuracy potentially offered by transmitting pre-processed samples (through the local convnet layer) to the centralized DNN is sufficient to amortize the network cost (or latency cost) to transmit this sample (with the cost tunable at will by the operator or slice owner).

Figure 52 demonstrates the potential scalability benefits that such a distributed implementation can offer compared to a fully centralized baseline. For example, focusing on the black curve (the different curves correspond to different weights balancing the strength of the local and remote layers of the DDNN, an interesting tradeoff in itself), we can make some very interesting observations:

- Compared to the leftmost values ("all predictions delegated to the stronger, centralized DNN"), and moving towards the right of these plots, we can see that 50% of predictions could be made locally with an inaccuracy cost increase of less than 2-3%, and 80% of predictions could be made locally with less than 5% accuracy degradation.
- In fact, the above quoted inaccuracies are with respect to a DDNN that has been trained with a local exit, but still performs all prediction at the remote/centralized exit. Compared to the *actual baseline* of a fully centralized DNN (e.g., DeepCog architecture used for MSE prediction [58], indicated in the two figures as a purple cross, a very interesting phenomenon occurs in some scenarios (e.g., the right plot): *50-60% of predictions can be resolved locally (and thus with much less latency and zero overhead for the network)* ***without any accuracy deterioration!*** The latter phenomenon is due to the performance benefits of simply having local exits during offline training even in a centralized DNN [60].

Summarizing, a key take home message of the current architecture is that, *up to a 10x traffic reduction can be achieved compared to the centralized baseline, with minimum performance degradation (less than 5% MSE increase)*, as promised in the MonB5G description of work. As decision latency and/or network traffic can often be the main bottlenecks in such AI-based setups, this 90-10 regime that can be supported suggests that 90% of the decisions related to these KPIs can also be acted on locally, by the collocated DE, hence greatly facilitating the distributed/parallelized operation at the DE level as well.

As the details of the offloading mechanism and DDNN training for composite KPIs have been mainly tested with SLAs related to resource allocation examples [61], we will further elaborate on those details in WP4 deliverables.

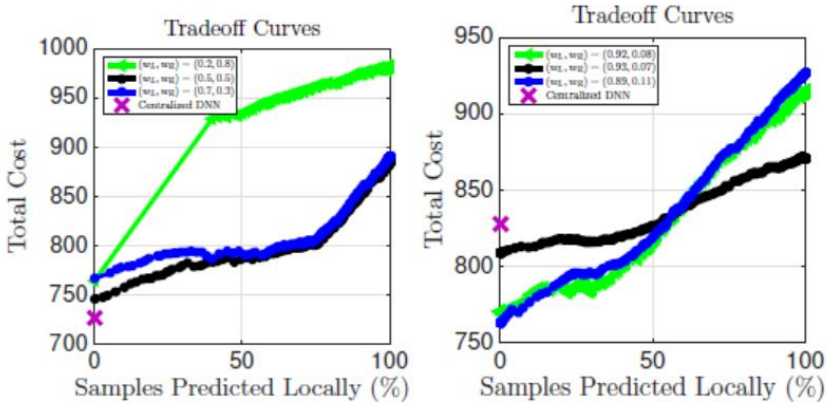


Figure 52 Scalability curves, demonstrating the tradeoff between cost from prediction inaccuracy, e.g., MSE, (y-axis) and percentage of predictions made locally (x-axis); The 3 curves represent different training weights for the offline tuning of the Distributed DNN, while the actual accuracy cost (y-axis) and network traffic (x-axis) are measured online, on test data. The cross corresponds to the baseline performance of a fully centralized DNN network with the exact same number of convolutional layers as the distributed one.

8 MonB5G Analytics Engine for Network Fault Management

This Chapter is devoted to implement the MonB5G AE features for Network Fault Management. Fault Management is a fundamental part of the FCAPS (Fault, Configuration, Accounting, Performance and Security) operations. It aims to detect and eliminate any malfunctions that have occurred in the monitored systems to prevent the degradation of the provided services. In general, the process of fault management can be dissected into three steps:

- Fault detection – the system malfunctions can usually be observed in the form of degradation of KPIs or abnormal behaviour of specific components. Therefore, the capability to efficiently detect network anomalies is of particular importance for the network operators;
- Fault diagnosis and isolation – the set of actions to identify the source of the fault and mitigation of its impact on the system's behaviour;
- Correlation and aggregation – gathering of events that correspond to the fault and performing root cause analysis;
- Service restoration – execution of actions to restore the services that the fault has impacted;
- Fault correction is the step comprised of taking actions needed to eliminate the fault.

Typically, fault management is performed in two ways: passive and active. The first one relies on the collection of the alarms generated by the components (e.g., using SNMP protocol in case of network hardware) and undertaking the necessary actions on their basis. The main issue with this approach lies in the malfunction reporting strategy. As the management relies solely on the data provided by the monitored component in case of complete malfunctions, the alerts might not be generated; hence the issues that occurred will not be detected by the management system. The active approach, on the other hand, involves monitoring as well as interaction with the system components. If the element fails to respond to the management system calls, the relevant alarms are triggered. Usually, the active approach involves monitoring system KPIs and using relevant system information (hardware specifics, history of faults etc.) to predict the possible future faults and mitigate them before they occur. The MonB5G framework can support both passive and active fault management.

In Section 8.1 we propose an LSTM-based approach to outlier detection that can predict normal traffic behaviour in the sample sequences and detect faults by identifying large deviations from the expected behaviour. Since a system fault is an event that should be categorized as an exception, in this section we provide more general algorithms to identify outliers in the system. Anomalies can be defined as an unseen pattern that does not match known patterns learned from normal data. The rising volume of network data due to the generation of approximately 1 million events every second in RAN has made intelligent monitoring of network performance management imperative for network operators. To ensure infrastructures provide a high level of robustness to customers, rule-based systems based on domain knowledge were implemented to analyse and detect the anomalies across multiple features. However, these traditional rule-based network application approaches fail when exposed to new previously unseen complex patterns. This has highlighted the importance of anomaly detection and has resulted in drawing the researcher's attention towards adapting autonomous and intelligent network systems.

In particular, deep learning algorithms are capable of processing and learning from large data sets efficiently. They can be applied to mine the very large data sets that are generated in large operator's networks and

future 5G networks, and to find insights and other useful discoveries. In Section 8.2 we propose a RNN based solution to design and develop neural network-based anomaly detection solutions with reduced training and testing times, and get optimal anomaly detection results.

Lastly, another important aspect of anomaly detection and fault management is the identification of nodes in a network that are not functioning properly. The deviation of these nodes from the standard expected performance (which makes them *outliers*) may be *due to* a local fault or may become *the cause* of a system fault in the short term. In any case, such outlier behaviour needs to be identified as soon as possible. However, carrying out this identification should not require the individual nodes to disclose their private data. To this end, in Section 8.3 we propose an algorithm for solving learning problems in a network that allows each node to compare its local state with the average state of the network and identify itself as an outlier, while keeping the nodes' data private.

8.1 Fault and Anomaly Detection Based on Link Traffic Observations with LSTM

This section presents data-driven, LSTM-based AE for anomaly detection in network traffic. The described AE operates on real data obtained from MS of the operator's backbone links and can be used. i.e., for anomalies detection fault management of network slices performing functions associated with the operation of Transport Network (TN).

8.1.1 CONCEPT

The high-level architecture of the developed engine is presented in Figure 53.

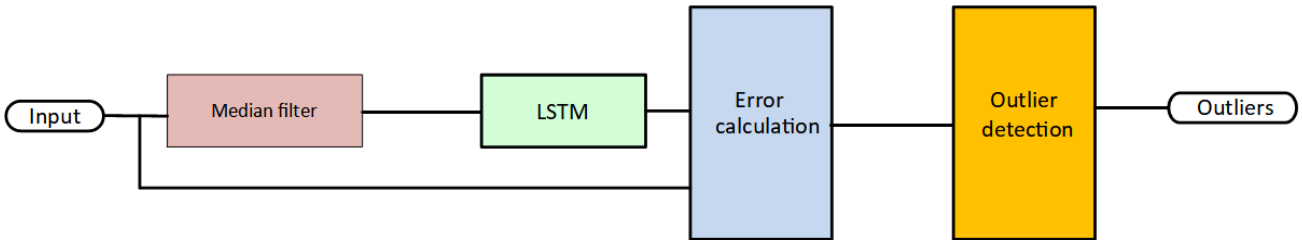


Figure 53 Architecture of LSTM-based AE for anomaly detection in TN slices.

The analysed data are link traffic samples that are acquired by the MS (IPFIX/Netflow probes) responsible for the monitoring of the backbone links. The obtained trace is filtered using median filtering to remove artefacts caused by the process of traffic samples acquisition - the exact accuracy is not essential to observe anomalies. The module employs one LSTM unit working in the AutoEncoder mode. The unit's role is to recreate the original, typical input curve without existing anomalies (perform smoothing of the curve in the selected historical period). The configuration of LSTM has been chosen based on the empirical results, providing satisfactory training error for the whole data-trace with good generalization and without overfitting. The network was trained with a learning rate equal to 0.01 and Adam [62] optimizer for the duration of about 200 epochs. For the training, we have used two training sets; each of them was five days long (1440 samples). The first training set was selected from the beginning of the trace; the second set was chosen from the final

part of the trace as we have noticed a significant change of the traffic. In both cases, the training set contained 1400 samples (c.a. 5 days), while the size of the testing set was the whole acquired trace (i.e. 90k samples). Both subsets are presented in Figure 55, marked as XTrain1 and XTrain2, respectively. The training data has been pre-preprocessed, which included data normalization (reshaping the data from the original range so that all values are within 0 and 1).

The derived LSTM structure is as follows:

- Input layer: 1 channel
- LSTM layer: 100 elements
- Fully connected layer: 100x1
- Output layer: 1 channel

The output of LSTM is fed to the module responsible for error calculation; it subtracts the original input signal from the LSTM output. The absolute value of error is later used within the outlier detection module for anomaly detection. To find the outliers the following approach that main goal is to find the background (envelope or amplitude) of the error.

The main goal of the AE is to distinguish anomalies based on the history of the past data (no future samples are taken into account). For this purpose, several approaches have been evaluated. One of the considered approaches included the removal of outliers that have values greater than three times of standard deviation, however it is sometimes not a suitable solution as an absolute error signal rarely follows the Gaussian distribution. Outliers can also be cut off based on mean calculations, but it often provides a high number of false positives. A promising solution for traces not having a non-Gaussian distribution was using the Inter Quartile Range (IQR). The IQR is calculated as the difference between the 75th and the 25th percentiles of the data. However, this method is susceptible to traces having a high variance making it inapplicable in this case. Other considered methods of finding outliers include k-nearest neighbour algorithm as an example of unsupervised learning, support-vector machines (SVM) and isolation forest.

To estimate the error signal envelope, a new approach has been proposed. This iterative algorithm consists of the following steps:

- Calculate the median of the local window, X (576 samples);
- Remove all samples that fulfil the condition $X(i) < median(X)$ and create a new vector Y
- Calculate the median of Y
- Remove samples (local outliers detection methods can be used such as Local Outlier Factor) above threshold (triple standard deviation, etc.)
- Go back to (1) until a condition (number of iterations, number of samples in the final set etc.) is met

The obtained envelope level is changing quickly, but in real implementation, the error threshold change is typically very slow – in some cases, there is even a fixed threshold. Due to this fact, a slowly varying threshold has been introduced. It has been done by limiting changes between two consecutive envelope values to a small, predefined value (step). During testing, satisfactory results have been obtained for the step size equal to 0.05% threshold change per step, i.e., about 15% threshold change per day. The obtained curve can be seen as a mask that separates anomalies from regular traffic. Typically, not an envelope is directly followed, but a certain margin (20%-50%) is added to avoid false alerts.

8.1.2 DATASET

The used dataset was acquired on a backbone link over a period of consecutive 340 days. The links' bandwidth was measured in 5-minute intervals. The exemplary unfiltered trace is presented in Figure 54a.

The observed traffic fluctuations show an exceedingly high periodicity. Typically, during the weekdays, the peak of activity can be observed at night (around midnight), while during the weekends, the behaviour is the opposite, and the highest activity occurs during the first half of the day. The sudden spikes in the observed traffic are caused by the operation of the measurement equipment and probing. Nonetheless, there exists a high correlation between the time of day (ToD) and the user activity. Such properties of the dataset enable building accurate models for prediction to assist anomaly detection mechanisms. In a result of the reconfiguration of the network or failure, the traffic changes its characteristics, it is "not identical" (not stationary) over the whole period.

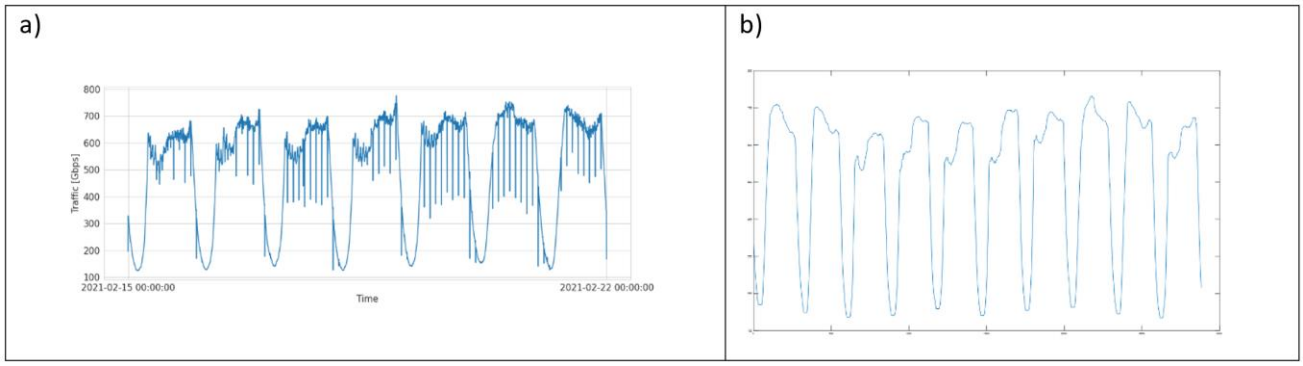


Figure 54 Traffic trace without pre-processing (a) and after performing median filtration (b)

As described in section 8.1.1, in order to eliminate anomalies caused by bandwidth probes, the median filtering was applied (Figure 54a). Figure 55 presents the full trace with marked anomalies that indicate potential faults.

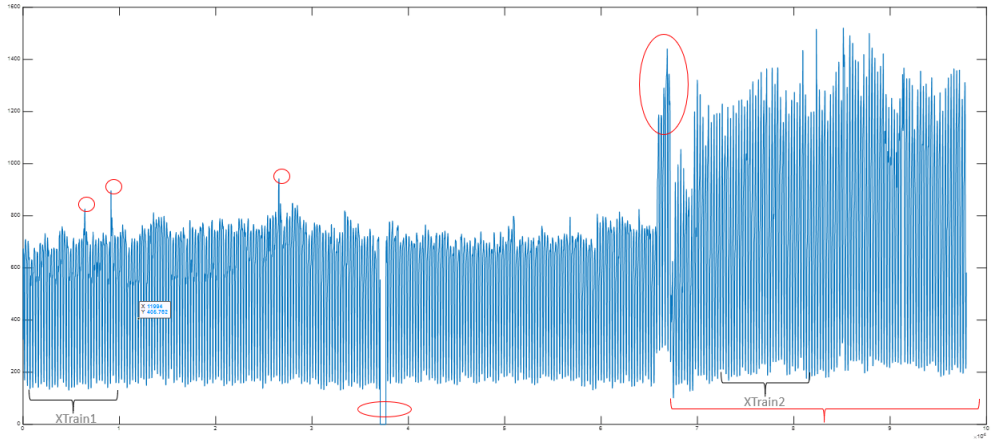


Figure 55 Full trace (340 days) with two data subsets used for training the LSTM network (Xtrain1 and XTrain2) with anomalies indicating potential faults (marked with red ellipses).

8.1.3 FAULT AND ANOMALY DETECTION RESULTS

In this section, the details regarding the results of LSTM-based anomaly detection AE will be described. The concept evaluation has been done using two separate datasets, XTrain1 and XTrain2, and tested against the whole 340-day trace, as presented in section 8.1.2. Figure 56 presents the results of performing the encode/decode operation using the XTrain1 subset as the input. The obtained YTrain1 set shows a very high resemblance to the original XTrain1 curve; however, small discrepancies can be observed in the areas where the slope changes rapidly.

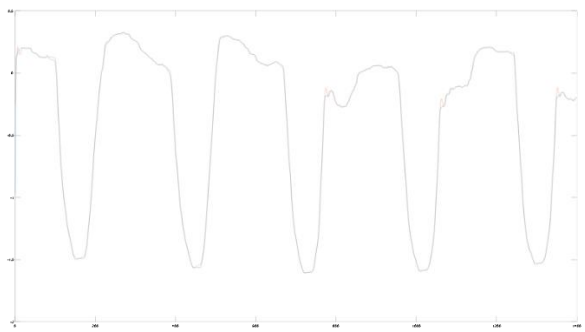


Figure 56 Training accuracy evaluation: original (blue) and trained (orange) time series

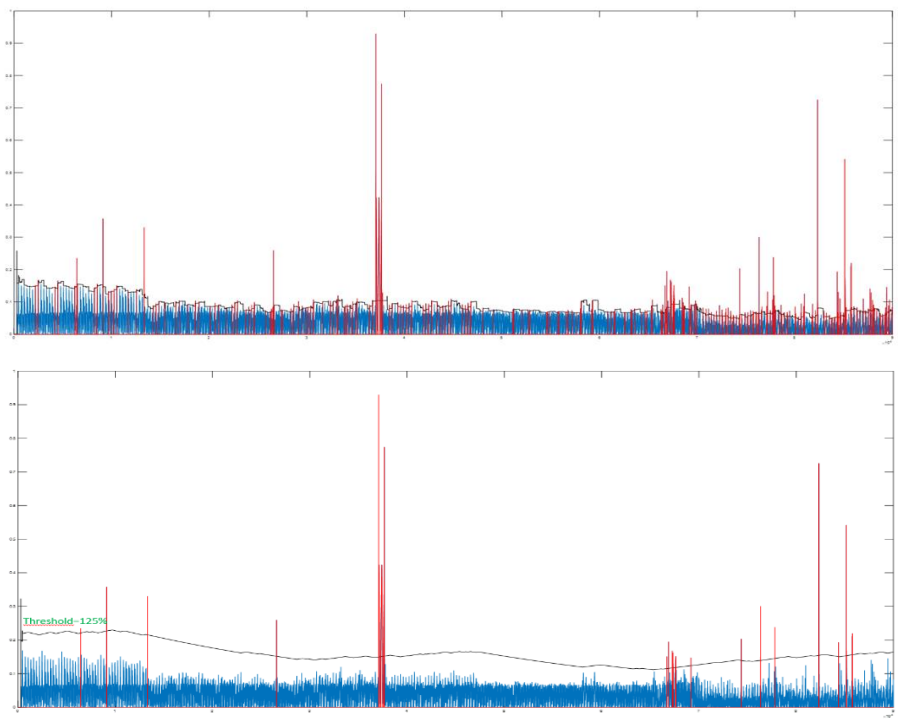


Figure 57 Anomalies detected with: (Top) the envelope estimation for Xtrain2; (Bottom) slowly varying threshold based on envelope of the top panel.

The trained LSTM unit has been used to encode the whole period of 340 days testing set. The obtained sequence has been afterwards subtracted from the original signal to get the error induced by the AutoEncoder operations. As the final step, the fault and anomaly detection algorithm described in Section 8.1.1 has been applied to detect the outliers in the obtained trace. For this purpose, several threshold values have been tested. The top panel of Figure 57 shows the anomalies detected using envelope estimation for the XTrain2 subset with the total detected outliers of 1798. The increase of the threshold by 25% caused the reduction of detected outliers to 922, which indicates the importance of the selection of the suitable threshold values (to minimise the number of false alarms and increase the anomaly detector accuracy). To optimise the anomaly detection process, hereby, we propose the second approach that uses a slowly varying threshold (Figure 57 bottom).

The obtained results are much more promising than in the case shown as Figure 57 top. In total, 774 events have been detected. Most of the anomalies are close to each other and form two separate groups. The analysis of the detected events with the prior information regarding faults has shown that 14 malfunctions (out of 14) have been detected. That includes two groups of consecutive anomalies (i.e. faults) and 12 isolated events (i.e. anomalies).

The presented approach is characterised by very high scalability. Each AE for anomaly detection analyses a single trace (link utilisation) and can be deployed for a single link, a set of intra-domain links, inter-domain links or the e2e link. The traffic aggregation, however, can severely impact the accuracy of fault detection as the singular anomalies can be stacked together and be observed as a normal system behaviour or local fluctuations (that can be removed in the pre-processing stage). Moreover, the implemented AE can be used for anomaly detection for any KPIs considering the fulfilment of the requirement regarding data dimensionality (the input is a vector).

8.2 Local Analytics Engine for Fault Management with RNN

In this section we present an alternative approach for time series analysis and anomaly detection based on recurrent neural networks (RNN).

This model architecture as shown in Figure 58 combines the AE predictions with the Network Fault Management to identify the faults (anomalies) to provide root cause analysis. The local analytic engine predictions used to detect faults are based on an unsupervised learning probabilistic prediction approach trained using GRUs.

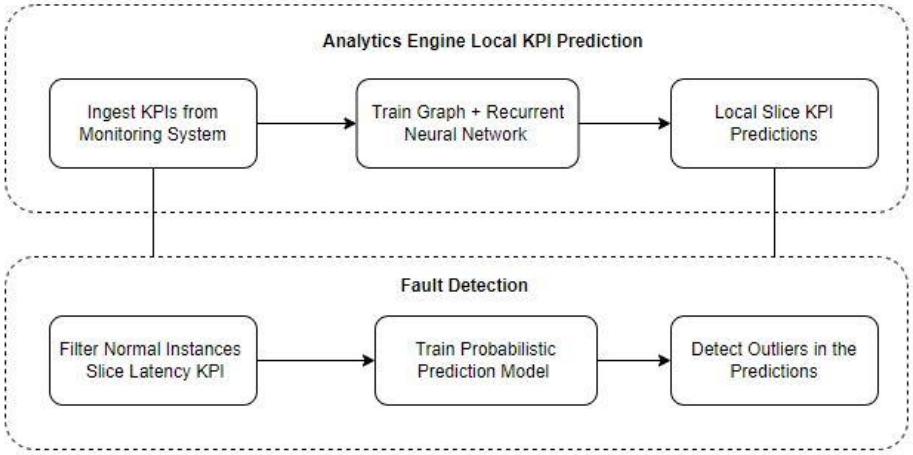


Figure 58 Analytics Engine with Fault Management Model

8.2.1 FAULT DETECTION MODEL

A language model for sequences specifies a probability distribution for the next call in a sequence given the sequence of previous system calls. The Gated Recurrent Unit Based Neural Network is trained to produce this probability distribution using a training set of known normal sequences, that is, the network learns a language model of normal sequences.

Given a set of sequences, we estimate the probability of a sequence occurring using probability distributions. Note that $p(x_i \mid x_{1:i-1})$ is the probability of the integer xi occurring after the sequence $x_{1:i-1}$.

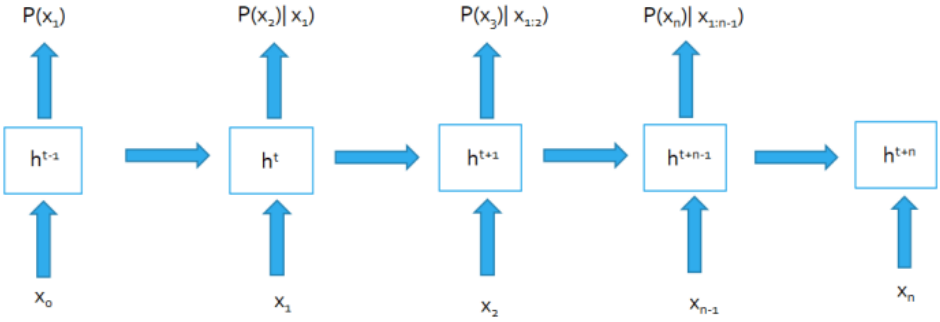


Figure 59 Conditional Probability over sequences

In practice, the negative log of the value p(x) defined in equation above is used resulting in high values for unlikely sequences and low values for likely sequences. Anomaly detection for sequences can be carried out by imposing a threshold for this negative log likelihood (L) and predicting an anomaly for sequences with an L value above this threshold.

8.2.2 ALGORITHM FOR PREDICTION BASED ANOMALY DETECTION

Given a set of normal sequences, the anomaly detection algorithm evaluates the test slice latency as anomalous or normal. As the algorithm implemented is cost efficient, it consumes much less computational resources in comparison to the traditional deep neural network algorithms. In this section, we discuss the pseudo code for anomaly detection.

Training

- The RNN based model is trained in mini batches on normal sequences in the slice latency time series data (no anomalous sequences).
- The model is trained in mini batches with the set of normal sequences (no anomalous sequences). The loss function is the categorical cross entropy function.

Calculation of Sequence Probability

- The input sequence is fed through the trained model. The output is a sequence of probability distributions using hidden states, which represent the probability distribution for the next integer in the sequence.

$$p(x) = \prod_{i=1}^{l} p(x_i | x_{1:i-1})$$

- A sequence probability value is calculated by essentially multiplying the probabilities of the next integer in the sequence occurring, across the entire length of the sequence.

Calculation of ROC curve data and AUC for validation data

- The negative log of the sequence probability is calculated for every sequence in the validation data.
- If the negative log value for the sequence is greater than the threshold, the sequence is classified as anomalous (Positive), otherwise it is classified as normal (Negative). (This corresponds to the sequence probability for abnormal classification being below a threshold, i.e., unlikely).
- The sequence is identified as either True Positive, False Positive, True Negative, False Negative.
- The Detection Rate and False Alarm Rate for the particular threshold value is calculated.
- The ROC curve is plotted for the range of threshold values.
- The AUC value is calculated for the ROC curve.

8.3 Outlier Identification in Networks with Decentralized Optimization

We have already remarked the importance of data-driven anomaly detection and managing solutions that achieve smarter decisions with less human intervention. When such tasks are executed in a network where all nodes both gather data and need to be surveilled to detect faults in the system, the size of the network imposes a hard challenge. Indeed, as the size of the network increases, so does the amount of data that need to be processed, which makes traditional centralized management solutions unsuitable due to communication overhead, stringent latency requirements, and privacy concerns.

In contrast, fully-decentralized solutions have the potential to alleviate all of these issues at once. However, decentralized solutions come with their own challenges, namely (i) designing decentralized algorithms with comparable (and ideally, matching) performance to that of their centralized counterparts, (ii) dealing with

potentially very different devices in terms of communication and computation capabilities, and (iii) achieving the above without compromising the privacy advantages of the decentralized solution.

In the centralized learning setting, each node i sends its data $\{X_j\}$ for $j = 1, \dots, N_i$ and $i = 1, \dots, n$ to a centralized server that solves the problem

$$\underset{\theta \in \mathbb{R}^d}{minimize} \sum_{j=1}^{N} F(\theta; X_j)$$

For $N = N_1 + \cdots + N_n$, where the function $F(\cdot)$ is a cost that depends both on the data and the model parameter θ . The objective is thus finding the optimal value $\overline{\theta}$ that minimizes the sum of the costs over all data in the network. This general problem can capture many network management problems such as resource allocation and user association. However, the centralized solution has a great drawback: it needs all nodes to send their data to the central location. This incurs a huge communication overhead that may have to be paid multiple times if data is continuously generated at all nodes. Furthermore, it requires sacrificing privacy, since once the data leaves the node the control on its usage is lost. As a consequence, we assume that this approach is non-viable.

On the other hand, decentralized algorithms solve the problem

$$\underset{\theta \in \mathbb{R}^d}{minimize} \sum_{i=1}^{n} f_i(\theta)$$

where

$$f_i(\theta) = \sum_{j=1}^{N_i} F(\theta; X_j)$$

are private functions known only by each node $i = 1, \dots, n$ and depend on the data gathered at the node. Decentralized algorithms to solve this problem are able to find the same solution as the centralized algorithm but exchanging optimization variables (parameters, gradients) instead of the data itself, which lowers the communication costs and preserves privacy. Algorithms to solve the decentralized optimization or similar problems have been successfully applied to problems in sensor networks [63] [64], edge computing [65] [66], and network slicing [67] [68].

However, some of the most successful algorithms in the literature that tackle the decentralized problem are *synchronous*, i.e. they require some central coordinator that ensures that all nodes have completed an iteration to start the next. For very large networks, such an approach necessarily incurs in large communication costs and increased wall-clock time, since nodes who have quickly completed an iteration must wait for those that have not finished yet (see section Scalability of Asynchronous Algorithms below).

To alleviate the problem, a large body of literature has proposed asynchronous algorithms where nodes can activate at any time and communicate with one of their neighbors to complete an iteration of the algorithm without being managed by a global synchronization enforcer [69] [70] [71]. However, most of the existing approaches select the neighbor to contact based on a fixed probability (e.g., uniformly at random), a choice that ignores the optimization landscape at the moment of activation. We thus propose a solution that takes

into account this information to achieve faster convergence in terms of number of iterations than the uniform sampling baseline.

8.3.1 CONTRIBUTIONS

We propose an algorithm to solve the decentralized optimization problem that has the following convenient characteristics:

- It is completely decentralized, in the sense that all nodes in the network can obtain the global minimum value $\overline{\theta}$ communicating only with their neighbors, without the need of a central coordinator.
- It is asynchronous, in the sense that nodes can activate at any time and talk with their neighbors without having to wait for any other specific event in the network.
- Nodes can solve the decentralized optimization problem without having to share their own private data.

The key feature of our algorithm respect to existing approaches is that it considers the improvement provided by all its neighbors to choose the neighbor to contact to complete an iteration. For this, we propose to use the Dual Ascent (DA) algorithm [72] to solve a constrained reformulation of the decentralized problem showed above. We then consider three variants of this Decentralized DA (DDA) algorithm, whose differences are illustrated in Figure 60:

- **Uniform Sampling (DDA-US):** When a node activates, it chooses the neighbor to contact uniformly at random.
- **Maximum Gradient (DDA-MG):** When a node activates, it chooses the neighbor that corresponds to the dual parameter with the largest gradient, and thus, the one that provides the largest cost improvement.
- **Maximum Stored Gradient (our proposed solution, DDA-MSG):** Nodes keep in memory the magnitude of the last gradient applied with each of their neighbors and each time they activate, they choose the neighbor whose *stored* gradient is the largest.

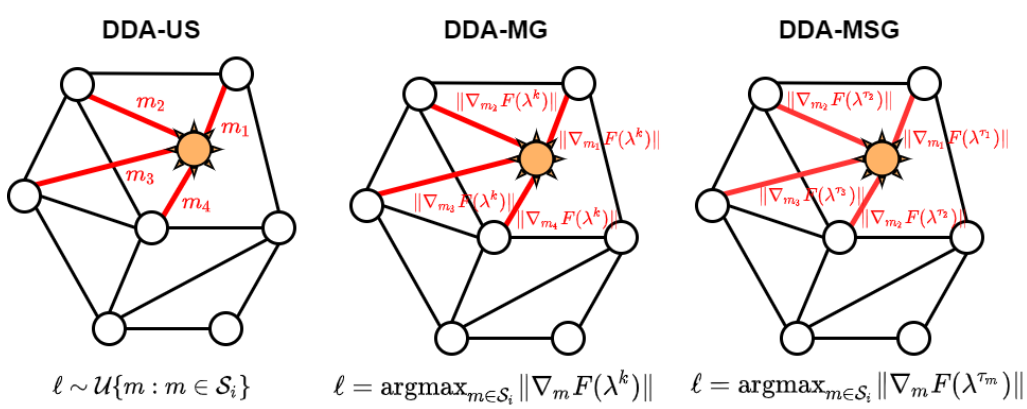


Figure 60 Differences in the neighbor choosing criterion of the algorithms considered

Although DDA-MG provides a significant improvement respect to DDA-US, it has also a remarkably larger cost in terms of computation and communication. In contrast, our proposed algorithm DDA-MSG approximates the performance of DDA-MG at the communication and computation costs of DDA-US, as shown in section Outlier Detection and Fault Management.

8.3.2 SCALABILITY OF ASYNCHRONOUS ALGORITHMS

Here we illustrate how asynchronous decentralized algorithms are less affected by the increase in the network size with respect to synchronous algorithms. Figure 61 shows a scheme of how the nodes might spend their time in each case.

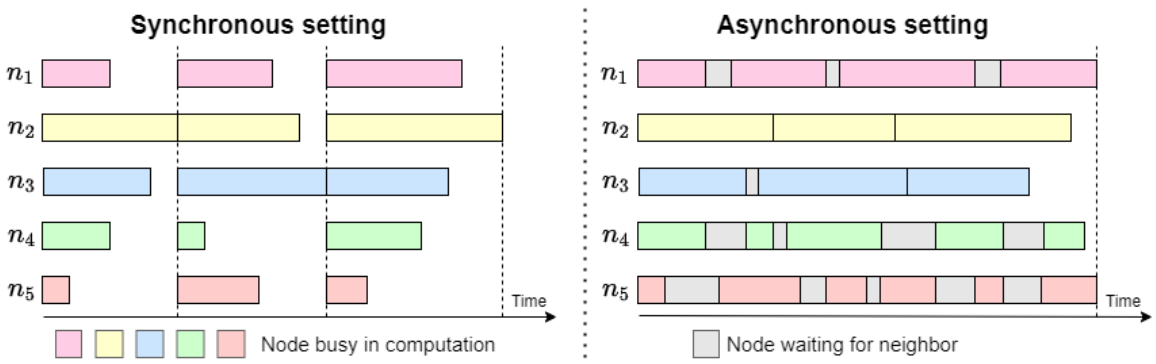


Figure 61 Time spent by each node in the synchronous and asynchronous settings

The left panel of Figure 61 illustrates how in the asynchronous setting nodes that can complete computations faster (in the example, nodes 1, 4 and 5) stay idle a large portion of the time waiting for the slower nodes (2 and 3). In contrast, in the asynchronous setting the fast nodes have the possibility of contacting different neighbors and spend much more time active than in the synchronous case.

To further illustrate these differences, we ran DDA-US versus synchronous dual ascent for a case where the $f_i(\theta)$are quadratic functions with $\theta \in \mathbb{R}^5$, and we repeated the experiment for random graphs of sizes $n = 10, 20, 30$, shown in Figure 62.

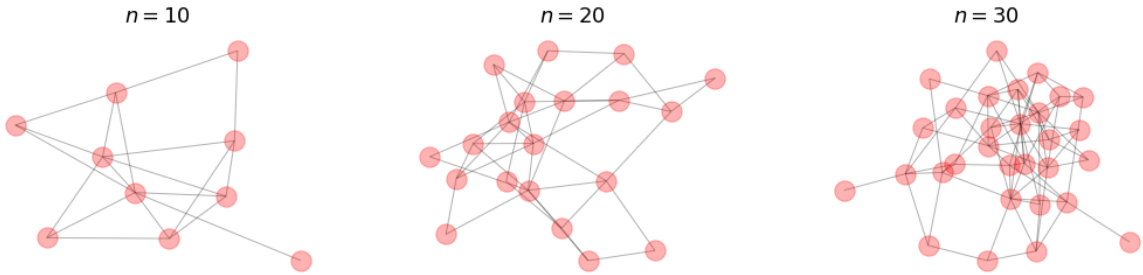


Figure 62 Graphs used in the comparison of synchronous versus asynchronous decentralized dual ascent

In our simulation we assumed that nodes wake up following an exponential distribution with rate $\lambda_i = \frac{1}{\beta_i}$ and the β_i are sampled from a beta distribution with parameters $\alpha = 2$ and $\beta = 50$. Figure 63 shows the histograms of the sampled β_i for each network.

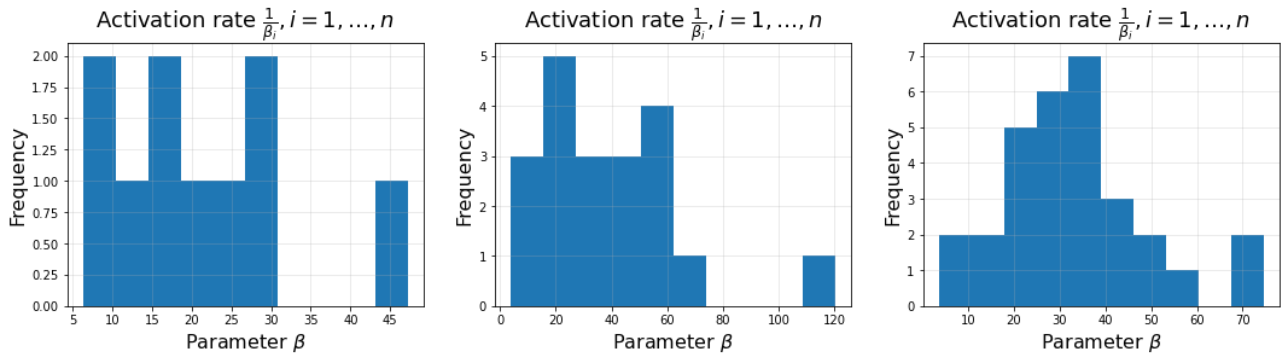


Figure 63 Histograms of the generated for the comparison of the synchronous versus the asynchronous algorithms

Figure 64 shows the convergence of the synchronous and asynchronous algorithms in all three cases. The asynchronous algorithm is faster than the synchronous for all three graphs, but the difference in time between the two for reaching a sub-optimality of 10e-4 increases with the number of nodes in the network. This displays how the advantages of asynchronous with respect to synchronous algorithms become larger as the network size increases.

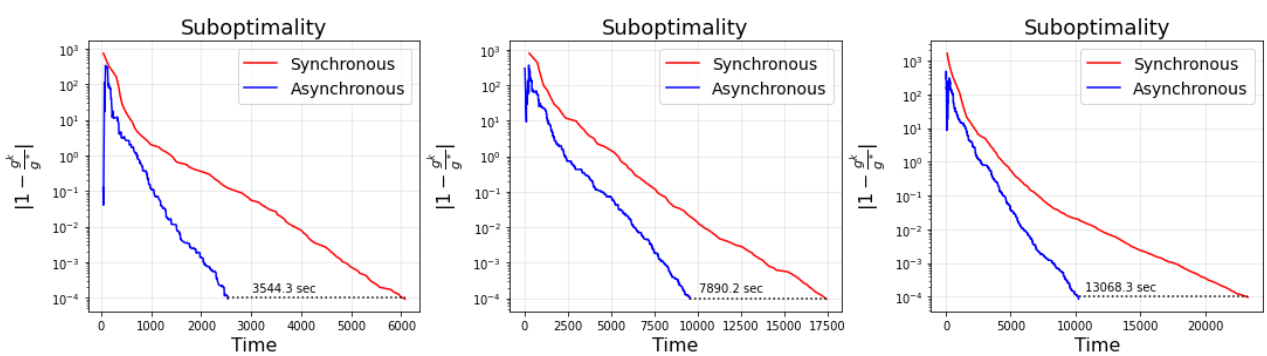


Figure 64 Simulation of random exponential node activation. The plots show the convergence achieved by the synchronous and asynchronous decentralized dual ascent algorithms for graphs with 10, 20 and 30 nodes respectively.

8.3.3 OUTLIER DETECTION AND FAULT MANAGEMENT

To display the performance of our algorithm DDA-MSG, we consider a scenario where base stations (BSs) in a network attempt to learn the global traffic in the network and identify potential load imbalances, which allows then to change the user association or reallocate resources online if one BS in the network finds more traffic that in can cope with. By letting nodes know the average state of the network and its fluctuations,

nodes can know if the overloaded BS is an outlier and the problem can be handled by reallocating resources, or if simply all BSs are equally loaded and there should be no feasible solution to tackle the problem with reallocation. Detection of such imbalances can also be used to identify times of the day when a BS is particularly under-loaded (resp. overloaded), case in which it may increase (resp. try to handover) its load to improve the overall performance of the network (e.g. through calendaring [73] or cell breathing techniques [74]).

In our simulations we assumed that the BSs are geographically distributed following a Poisson Point Process, as is commonly done in analyses of cellular networks, especially ones with small cells [75]; the resulting network is shown in Figure 65. Figure 66 shows the relative error respect to the optimum at each iteration for three asynchronous decentralized algorithms: (i) DDA-US, which samples the neighbor to uniformly at random, (ii) DDA-MG, which has higher communication and computation costs than DDA-US but also faster per-iteration convergence, and (iii) our proposed algorithm DDA-MSG, which approximates the performance of DDA-MG at the cost of DDA-US using past gradient information that is readily available at no extra cost. Our algorithm DDA-MSG performs closely to the fast (but costly) algorithm DDA-MG, but it runs at the cost of the cheap (but slow) DDA-US algorithm.

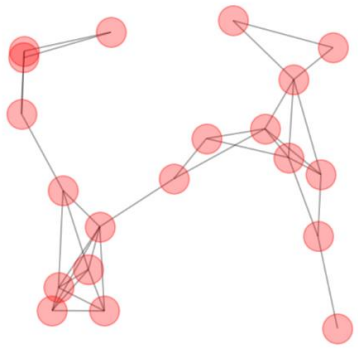


Figure 65 Geographic graph showing the connectivity between base stations used in our simulations

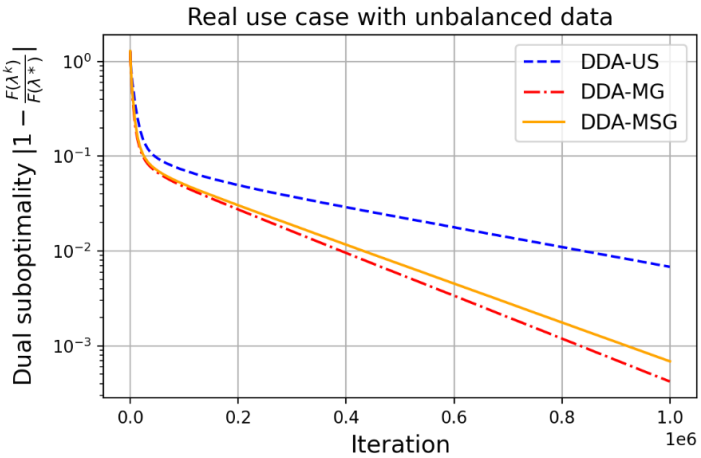


Figure 66 Convergence of the three decentralized dual ascent (DDA) algorithms considered: uniform neighbor sampling (DDA-US), maximum gradient neighbor selection (DDA-MG), and our proposal, stored maximum gradient neighbor selection (DDA-MSG)

Lastly, Figure 67 illustrates how all nodes, independently of their position in the network, converge to the global model, even though local data significantly differ. The yellow lines show the data of BSs $i = 2, 7, 11$ and the red lines show the fits of their local models. These three base stations differ largely in their daily traffic profiles, but despite this high heterogeneity, the solutions retrieved by DDA-MSG at each BS (black dashed lines) are practically identical, and in all cases very different to the local solutions (red lines). This confirms that the solutions retrieved by our algorithm are the optimal value $\overline{\theta}$ of the decentralized optimization problem and are not biased by the local data at each node.

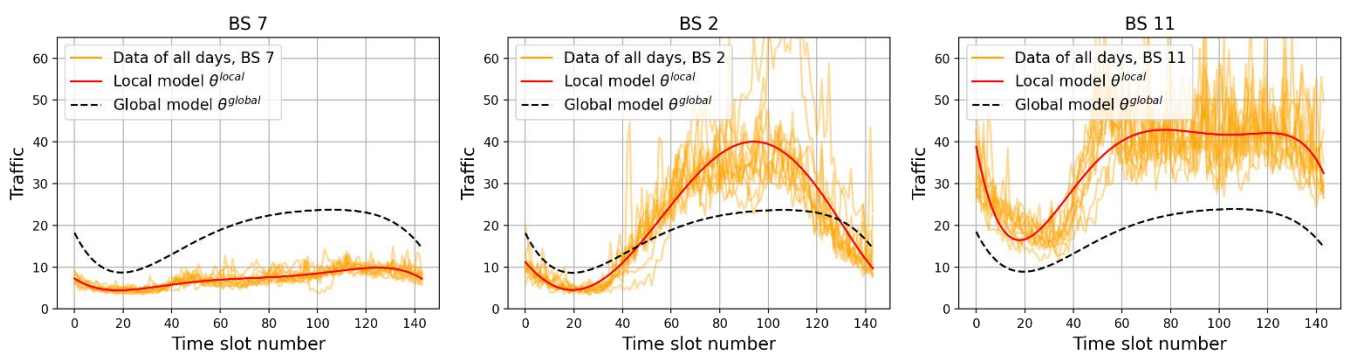


Figure 67 Data heterogeneity between base stations 2, 7 and 11, and comparison of the local and the global parameter fits. The figures show the data of 3 base stations in the network (yellow lines) with the fit of their local data only (red line) and the fit of their local copy of the global parameter (black dashed line).

9 Conclusions

This deliverable presented our achievements in WP3 of the MonB5G project: *the distributed AI-driven MS and AE for scalable zero-touch slice management*. In addition, two synthetic 5G datasets were reported to benefit the 5G/B5G research community for development of new AI-driven solutions in the context of network data analytics. MonB5G MS and AE embed themselves in a decentralized management architecture to distribute monitoring and analytics among various layers, nodes and sub-slices of the management hierarchy, such that the management tasks are executed locally for reducing communication overhead and delay. The current tests have demonstrated the advantages of the proposed techniques with promising results on scalable slice management.

MS provides up-to-date monitoring for online analysis and reconfiguration in response to unexpected network dynamics. In detail, it presents the following innovations. First, MS is designed based on the microservice architecture and implemented as a cloud-native application in e.g., a Kubernetes cluster. Next, MS is under a hierarchical architecture, which allows multiple sub-MSs which are collaborating in a master-slave pattern, and the data monitored by the slave MS can be further gathered by the master MS as needed. Moreover, MS reduces the footprint of the system. MS entities can be shared among administrative components i.e., AE and DE. Last but not least, each sampling function, used by MS to collect data, is implemented by a single Docker image. In this way, the sampling loop resides inside the Docker image, and there is no need to frequently create and destroy the sampling function container. All these techniques enable MS to track the current status of a large number of slices for scalable slice management.

AE is also structured hierarchically, and empowered by advances of distributed AI techniques, which were tailored and extended to provide hierarchical KPI prediction and fault management on different management layers. To enable online prediction, analytics tasks will be distributed among adjacent layers vertically and horizontally with optimal policies learned via distributed ML and FL techniques. The developed novel techniques are able to balance the prediction performance and communication overhead/delay. AE provides a variety of features, such as: context-aware traffic prediction, network slicing resource provisioning with FL, network-aware KPI prediction with distributed neural networks, local fault identification with RNN, and outlier detection with decentralized optimization.

In a summary, we developed decentralized MS/AE, alone with distributed AI techniques. The achievements reported in this deliverable demonstrated that this work is a decisive step towards scalable and automatic slice management.

References

[1] S. Sharma, R. Miller and A. Francini, "A Cloud-Native Approach to 5G Network Slicing," *IEEE Communications Magazine,* vol. 55, no. 8, pp. 120-127, 2017.

[2] I. P. Chochliouros, A. S. Spiliopoulou, P. Lazaridis, A. Dardamanis, Z. Zaharis and A. Kostopoulos, "Dynamic Network Slicing: Challenges and Opportunities," in *Proceedings of AIAI-2020, IFIP Advances in Information and Communication Technology*, 2020.

[3] MonB5G, "Deliverable D3.1: Techno-economic analysis of the beyond 5G environment, use case requirements and KPIs," 2020.

[4] 5. PPP, "Cloud Native and 5G Verticals services 2020," 2020. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2020/02/5G-PPP-SN-WG-5G-and-Cloud-Native.pdf.

[5] U. Paul, J. Liu, S. Troia, O. Falowo and G. Maier, "Traffic Profile and Machine Learning Based Regional Data Center Design and Operation for 5G Network," *Journal of Communications and Networks,* vol. 21, no. 6, pp. 569-583, 2019.

[6] A. Dalgkitsis, M. Louta and G. T. Karetsos, "Traffic Forecasting in Cellular Networks Using the LSTM RNN," in *Proceedings of the 22nd Pan-Hellenic Conference on Informatics*, 2018.

[7] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul and P. Bertin, "Improving Traffic Forecasting for 5G Core Network Scalability: A Machine Learning Approach," *IEEE Network,* vol. 32, no. 6, pp. 42-49, 2018.

[8] J. Wang, J. Tang, Z. Xu, Y. Wang, G. Xue, X. Zhang and D. Yang, "Spatio Temporal Modeling and Prediction in Cellular Networks: A Big Data Enabled Deep Learning Approach," in *Proceeding of IEEE Conference on Computer Communications*, 2017.

[9] A. Adebiyi, A. Adewumi and C. Ayo, "Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction," *J. Appl. Math.,* pp. 1-7, 2014.

[10] C. Nichiforov, I. Stamatescu, I. Fagarasan and G. Stamatescu, "Energy Consumption Forecasting Using ARIMA and Neural Network Models," in *Proceeding of the 5th International Symposium on Electrical and Electronics Engineering (ISEEE)*, 2017.

[11] S. Siami-Namini, N. Tavakoli and A. S. Namin, "A Comparison of ARIMA and LSTM in Forecasting Time Series,"," in *Proceeding of the 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018.

[12] T. Zhang and S. Mao, "Machine Learning for End-to-End Congestion Control," *IEEE Communications Magazine,* vol. 58, no. 6, pp. 52-57, 2020.

[13] C. Gutterman, E. Grinshpun, S. Sharma and G. Zussman, "RAN Resource Usage Prediction for a 5G Slice Broker," in *Proceedings of the 20th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019.

[14] S. Irina, S. Irina and M. Anastasiya, "Forecasting 5G Network Multimedia Traffic Characteristics," in *Proceeding of IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 2020.

[15] Y. Zhou, Z. M. Fadlullah, B. Mao and N. Kato, "A Deep-Learning-Based Radio Resource Assignment Technique for 5G Ultra Dense Networks," *IEEE Network,* vol. 32, no. 6, pp. 28-34, 2018.

[16] I. Alawe, Y. Hadjadj-Aoul, A. Ksentinit, P. Bertin, C. Viho and D. Darche, "An Efficient and Lightweight Load Forecasting for Proactive Scaling in 5G Mobile Networks," in *Proceeding of IEEE Conference on Standards for Communications and Networking (CSCN)*, 2018.

[17] A. Mozo, J. L. Lopez-Presa and A. FernandezAnta, "A Distributed and Quiescent Max-Min Fair Algorithm for Network Congestion Control," *Expert Systems with Applications,* vol. 91, pp. 492-512, 2018.

[18] R. Xie, X. Jia and K. Wu, "Adaptive Online Decision Method for Initial Congestion Window in 5G Mobile Edge Computing Using Deep Reinforcement Learning," *IEEE Journal on Selected Areas in Communications,* vol. 38, no. 2, pp. 389-403, 2020.

[19] T. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," arXiv preprint arXiv:1609.02907, 2016.

[20] M. Schlichtkrull, "Modeling Relational Data with Graph Convolutional Networks," in *Proceeding of European Semantic Web Conference*, 2018.

[21] Z. Yan et al., "Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach With Graph Convolutional Networks," *IEEE Journal on Selected Areas in Communications,* pp. 1040-1057, 2020.

[22] V. Mnih, "Asynchronous Methods for Deep Reinforcement Learning," arXiv: 1602.01783, 2016.

[23] M. Dolati et al., "DeepViNE: Virtual Network Embedding with Deep Reinforcement Learning," in *Proceeding of IEEE Conference on Computer Communications Workshops*, 2019.

[24] T. Li, A. K. Sahu, A. Talwalkar and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," *IEEE Signal Processing Magazine,* vol. 37, no. 3, pp. 50-60, 2020.

[25] J. Konecny, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," arXiv preprint arXiv:1610.05492, 2016.

[26] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar and V. Smith, "Federated Optimization in Heterogeneous Networks," in *Proceedings of Machine Learning and Systems*, 2020.

[27] V. Smith, C. K. Chiang, M. Sanjabi and A. S. Talwalkar, "Federated Multi-Task Learning," in *Advances in Neural Information Processing Systems*, 2017.

[28] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.

[29] H. Wang et al., "Data-Driven Dynamic Resource Scheduling for Network Slicing: A Deep Reinforcement Learning Approach," Information Sciences, 2019.

[30] Y. Xiao et al., "NFVdeep: Adaptive Online Service Function Chain Deployment with Deep Reinforcement Learning," in *Proceeding of IEEE/ACM 27th International Symposium on Quality of Service*, 2019.

[31] 3GPP, "System Architecture for the 5G System (5GC)," 3GPP TS 29.520 v17.1.0, Dec 2020.

[32] 3GPP, "Architecture Enhancements for 5G System (5GC) to Support Network Data Analytics Services," 3GPP TS 23.288 v16.6.0, Dec 2020.

[33] 3GPP, "5G System; Network Data Analytics Services; Stage 3," 3GPP TS 29.520 v17.1.0, Dec 2020.

[34] 3GPP, "Policy and Charging Control Framework for the 5G System (5GC); Stage 2," 3GPP TS 23.503 v16.7.0, Dec 2020.

[35] ETSI, "Zero-Touch Network and Service Management (ZSM); Closed-loop automation; Enablers," ETSI GS ZSM 009-1 V0.10.5, Jan 2021.

[36] N. J. Gunther, "A Simple Capacity Model of Massively Parallel Transaction Systems," in *Proceeding of CMG National Conference*, 1993.

[37] H. Hamman, "Superlinear Scalability in Parallel Computing and Multi-Robot Systems: Shared resources, Collaboration and Network Topology," in *Lecture Notes in Computer Science*, 2018.

[38] MonB5G, "Deliverable D2.4: Final release of MonB5G architecture (including security)," Oct. 2021.

[39] S. Knight, H. Nguyen, N. Falkner, R. Bowden and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications,* vol. 29, no. 9, pp. 1765-1775, 2011.

[40] A. Dalgkitsis, P. V. Mekikis, A. Antonopoulos, G. Kormentzas and C. Verikoukis, "Dynamic Resource Aware VNF Placement with Deep Reinforcement Learning for 5G Networks," in *Proceeding of 2020 IEEE Global Communications Conference*, 2020.

[41] A. Dalgkitsis, L. Garrido, K. Ramantas, L. Alonso and C. Verikoukis, "Schema: Service Chain Elastic Management with Distributed Reinforcement Learning," in *Proceeding of 2021 IEEE Global Communications Conference*, 2021.

[42] M. Peuster et al., "Medicine: Rapid Prototyping of Production-Ready Network Services in Multi-Pop Environments," in *Proceeding of IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

[43] B. Lantz et al., "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," 2010.

[44] M. Polverini, J. Galan-Jimenez, F. Lavacca, A. Cianfrani and V. Eramo, "Improving Dynamic Service Function Chaining Classification in NFV/SDN Networks through the Offloading Concept," *Computer Networks,* vol. 182, 2020.

[45] A. Martin, J. Egana, J. Florez, J. Montalban, I. G. Olaizola, M. Quartulli, R. Viola and M. Zorrilla, "Network Resource Allocation System for qoe-aware Delivery of Media Services in 5G Networks," *IEEE Transactions on Broadcasting,* vol. 64, no. 2, pp. 561-574, 2018.

[46] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "Openai GYM," arXiv preprint arXiv:1606.01540, 2016.

[47] P. Kairouz, H. B. McMahan and e. al., "Advances and Open Problems in Federated Learning," *Now Foundations and Trends,* 2021.

[48] A. Cotter et al., "Two-Player Games for Efficient Non-Convex Constrained Optimization and Other Data-Dependent Constraints," 2018. [Online]. Available: https://arxiv.org/abs/1804.06500.

[49] H. Chergui, L. Blanco and C. Verikoukis, "Statistical Federated Learning for Beyond 5G SLA-Constrained RAN Slicing," *IEEE Transactions on Wireless Communications,* vol. 21, no. 3, pp. 2066-2076, 2022.

[50] S. Roy, H. Chergui, L. Sanabria-Russo and C. Verikoukis, "A Cloud Native SLA-Driven Stochastic Federated Learning Policy for 6G Zero-Touch Network Slicing," in *Proceeding of IEEE ICC 2022*, 2022.

[51] J. X. Salvat et al., "Overbooking Network Slices through Yield-Driven End-to-End Orchestration," in *Proceeding of the 14th Intl. Conf. on Emerging Networking EXperiments and Technologies*, 2018.

[52] O.-L. J. et al., "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine,* vol. 55, no. 5, pp. 80-87, 2017.

[53] L. Le et al., "Applying Big Data, Machine Learning, and SDN/NFV to 5G Traffic Clustering, Forecasting, and Management," in *Proceeding of the 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018.

[54] A. Borghesi, B. Federico and M. Milano, "Improving Deep Learning Models via Constraint-Based Domain Knowledge: a Brief Survey," ArXiv abs/2005.10691, 2020.

[55] N. Muralidhar et al., "Incorporating Prior Domain Knowledge into Deep Neural Networks," Proceeding of IEEE Intl. Conf. on Big Data, 2018.

[56] P. Martinez-Julia, V. P. Kafle and H. Asaeda, "Using the Total Cost of Ownership to Decide Resource Adjustment in Virtual Networks," in *Proceeding of the 22nd Conf. on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2019.

[57] C. Zhang and P. Patras, "Long-Term Mobile Traffic Forecasting Using Deep Spatio-Temporal Neural Networks," in *Proceeding of the 18th ACM Intl. Symp. on Mobile Ad Hoc Networking and Computing* , 2018.

[58] D. Bega, M. Gramaglia, M. Fiore, A. Banchs and X. Costa-Perez, "DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning," in *Proceeding of IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.

[59] T. Italia, "Telecommunications - SMS, Call, Internet - MI," 2015. [Online]. Available: https://doi.org/10.7910/DVN/EGZHFV.

[60] S. Teerapittayanon, B. McDanel and H. Kung, "Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices," in *Proceeding of IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[61] T. Giannakas et al., "Fast and Accurate Edge Resource Scaling for 5G/6G Networks with Distributed Deep Neural Networks," in *Proceeding of IEEE WoWMoM 2022*, 2022.

[62] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimisation," [Online]. Available: https://arxiv.org/abs/1412.6980. [Accessed 20 04 2022].

[63] C. Fischione, "Fast-Lipschitz Optimization with Wireless Sensor Networks Applications," *IEEE Transactions on Automatic Control,* vol. 56, no. 10, pp. 2319-2331, 2011.

[64] S. Yang and J. A. Mccann, "Distributed Optimal Lexicographic Max-Min Rate Allocation in Solar-Powered Wireless Sensor Networks," *ACM Transactions on Sensor Networks (TOSN),* vol. 11, no. 1, pp. 1-35, 2014.

[65] B. Xiang, J. Elias, F. Martignon and E. Di Nitto, "Resource calendaring for Mobile Edge Computing: Centralized and decentralized optimization approaches.," *Computer Networks,* vol. 199, p. 108426, 2021.

[66] H. Chen, Y. Ye, M. Xiao, M. Skoglund and H. V. Poor, "Coded Stochastic ADMM for Decentralized Consensus Optimization With Edge Computing," *IEEE Internet of Things Journal,* vol. 8, no. 7, pp. 5360-5373, 2021.

[67] S. Doro, L. Bonati, F. Restuccia, M. Polese, M. Zorzi and T. Melodia, "Sl-EDGE: Network Slicing at the Edge," in *Proceeding of the 21st International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2020.

[68] A. Huang, Y. Li, Y. Xiao, X. Ge, S. Sun and H. C. Chao, "Distributed Resource Allocation for Network Slicing of Bandwidth and Computational Resource," in *Proceeding of IEEE International Conference on Communications (ICC)*, 2020.

[69] S. Pu, W. Shi, J. Xu and A. Nedic, "Push-Pull Gradient Methods for Distributed Optimization in Networks," *IEEE Trans. Automat. Control,* vol. 66, no. 1, pp. 1-16, 2020.

[70] E. Wei and A. Ozdaglar, "On the O(1/k) Convergence of Asynchronous Distributed Alternating Direction Method of Multipliers," in *Proceeding of 2013 IEEE Global Conference on Signal and Information Processing*, 2013.

[71] J. Xu, S. Zhu, Y. C. Soh and L. Xie, "Convergence of Asynchronous Distributed Gradient Methods Over Stochastic Networks," *IEEE Trans. Automat. Control,* vol. 63, no. 2, pp. 434-448, 2017.

[72] S. Boyd, N. Parikh and E. Chu, Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers, Now Publishers Inc, 2011.

[73] M. Morcos, J. Elias, F. Martignon, T. Chahed and L. Chen, "On Efficient Radio Resource Calendaring in Cloud Radio Access Network," *Computer Networks,* vol. 162, p. 106862, 2019.

[74] Y. Bejerano and S.-J. Han, "Cell Breathing Techniques for Load Balancing in Wireless LANs," *IEEE transactions on Mobile Computing,* vol. 8, no. 6, pp. 735-749, 2009.

[75] J. G. Andrews, F. Baccelli and R. K. Ganti, "A Tractable Approach to Coverage and Rate in Cellular Networks," *IEEE Transactions on communications,* vol. 59, no. 11, pp. 3122-3134, 2011.